

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

March, 2017

Table of Contents

Table of Contents	1
1. Brief description	2
2. Design Framework	3
2. Installation of Z-checker	4
3. Quick Start	4
3.1 Test Data Property Analysis	5
3.2 Test Calling R script from Z-checker	5
3.3 Test Compression Analysis	5
3.3.1 Simple assessment.....	5
3.3.2 Complete assessment	6
4. Application Programming Interface (API)	7
4.1 Initialization and finalization of the Z-checker environment.....	8
4.1.1 ZC_Init	8
4.1.2 ZC_Finalize.....	8
4.2 Generic I/O.....	8
4.2.1 ZC_readDoubleData in bytes.....	8
4.2.2 ZC_readFloatData in bytes	8
4.2.3 ZC_writeFloatData_inBytes	8
4.2.4 ZC_writeDoubleData_inBytes	8
4.2.5 ZC_writeData in text	8
4.3 Data property analysis	9
4.3.1 ZC_genProperties.....	9
4.3.2 ZC_printDataProperty	9
4.3.3 ZC_writeDataProperty	9
4.3.4 ZC_loadDataProperty	9
4.4 Data Comparison	9
4.4.1 ZC_compareData.....	9
4.4.2 ZC_printCompareResult	9
4.4.3 ZC_writeCompareResult.....	9

4.4.4 ZC_loadCompareResult	9
4.5 Setting monitoring calls in compressor codes	10
4.5.1 ZC_startCmpr	10
4.5.2 ZC_endCmpr	10
4.5.3 ZC_startDec.....	10
4.5.4 ZC_endDec.....	10
4.6 Calling R script from Z-checker	10
4.6.1 ZC_executeCmd_RfloatVector	10
4.6.2 ZC_executeCmd_RdoubleVector	10
4.6.3 ZC_executeCmd_RfloatMatrix.....	10
4.6.4 ZC_executeCmd_RdoubleMatrix.....	10
4.7 Plotting Z-checker analysis data	11
4.7.1 ZC_plotCompressionRatio.....	11
4.7.2 ZC_ plotHistogramResults	11
4.7.3 ZC_plotComparisonCases.....	11
4.7.4 ZC_plotAutoCorr_CompressError.....	11
4.7.5 ZC_plotAutoCorr_DataProperty	11
4.7.6 ZC_plotFFTAplitude_OriginalData	11
4.7.7 ZC_plotFFTAplitude_DecompressData	11
4.7.8 ZC_ plotErrDistribtion.....	11
4.8 Plot generic data by Gnuplot.....	12
4.8.1 genGnuplotScript_linespoints	12
4.8.2 genGnuplotScript_histogram	12
4.8.3 genGnuplotScript_lines.....	12
4.8.4 genGnuplotScript_fillsteps	12
5 Configuration file	12
6. Version history.....	12
7. Q&A and Trouble shooting	13

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 detect the compression quality (compression ratio, bit-rate), provide various global distortion analysis 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/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.

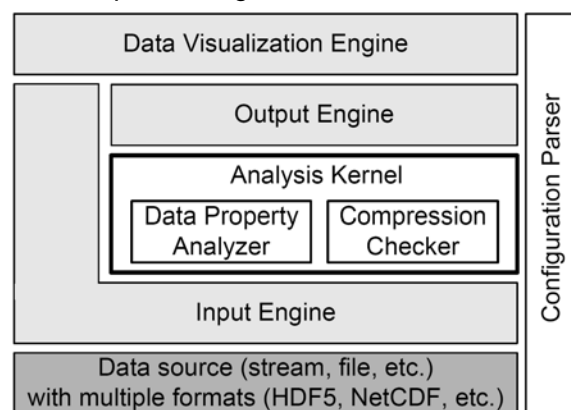


Fig. 1 Design Framework of Z-checker

- **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

The SZ software can be 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 and .a and .so libraries in [INSTALL_DIR]/lib

3. Quick Start

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 Test Data Property Analysis

Testing commands (in the examples/ directory):

analyzeDataProperty [config_file] [input_datafile] [dimension_size....]

Example: testdata/x86/testfloat_8_8_128.dat is a 3D variable (128x8x8 , where 128 is the slowest-changing dimension).

./analyzeDataProperty **zc.config** **testdata/x86/testfloat_8_8_128.dat** **8 8 128**

The data property analysis results will be put in the directory examples/dataProperties/.

[variable_name].prop file contains the property of the data, such as min**Value**, max**Value**, average**Value**, value_range, entropy, and auto-correlation.

[variable_name].autocorr keeps the auto-correlation results with different lags.

[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.

3.2 Test Calling R script from Z-checker

Testing commands (in the examples/ directory):

./testRscript_float **zc.config**

./testRscript_double **zc.config**

The above two testing examples are based on single-precision and double-precision data respectively. That is, they will convert the output of R script execution into single-precision data set and double-precision data set respectively. In the two examples, we use "cat R2.txt" and "cat R3.txt" to emulate the R script execution. In practice, you should replace them by your own command such as "Rscript r_script_file".

3.3 Test Compression Analysis

3.3.1 Simple assessment

If you just want to compare the reconstructed data with the original data, you can use **compareDataSets.c** in the examples/ directory, as shown below.

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

Example: **compareDataSets** **zc.config** **testfloat_8_8_128.dat** **testfloat_8_8_128.dat.out** **8 8 128**

Three output files (.cmp, .autocorr, and .dis) will be stored in examples/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.

3.3.2 Complete assessment

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 -xzf 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/libz.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_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_plotFFTAmlitude_OriginalData

```
void ZC_plotFFTAmlitude_OriginalData();
```

4.7.7 ZC_plotFFTAmlitude-DecompressData

```
void ZC_plotFFTAmlitude-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.

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>