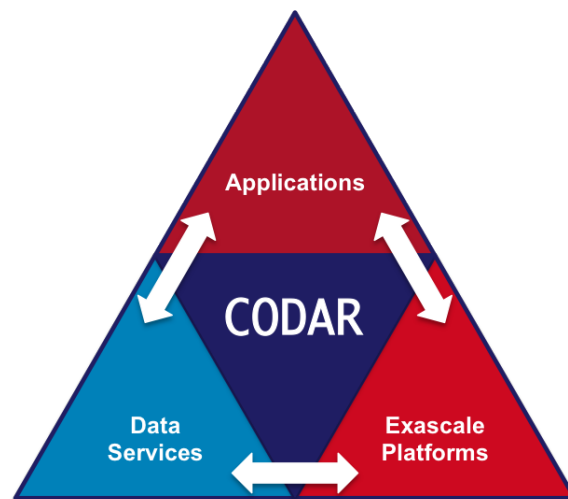


## CODAR Design Document

# Z-checker: a compression assessment tool

Version of January 17, 2018

Franck Cappello, Sheng Di



# Table of contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Stakeholders</b>	<b>1</b>
2.1	Exascale applications . . . . .	1
2.2	Software technologies . . . . .	2
2.3	Hardware technologies . . . . .	2
<b>3</b>	<b>Use cases</b>	<b>2</b>
3.1	Offline analysis . . . . .	2
3.2	Online analysis . . . . .	2
3.3	The ECP applications that have been assessed by Z-checker . . . . .	4
<b>4</b>	<b>Requirements</b>	<b>4</b>
<b>5</b>	<b>Related work</b>	<b>4</b>
<b>6</b>	<b>Design</b>	<b>5</b>
6.1	Overview . . . . .	5
6.2	Terminology . . . . .	6
6.3	Dependencies . . . . .	7
6.4	Z-checker command line format and basic usage . . . . .	8
6.5	Z-checker configuration files . . . . .	9
6.6	output directory layout . . . . .	12
6.7	One-command installation and usage . . . . .	13
6.8	A typical test plan . . . . .	13
6.9	Release Goals . . . . .	14
<b>7</b>	<b>Metrics</b>	<b>14</b>
<b>8</b>	<b>Work plan</b>	<b>14</b>
<b>9</b>	<b>Open questions</b>	<b>14</b>

# Z-checker: a compression assessment tool

Franck Cappello, Sheng Di

## 1 Introduction

---

Because of the 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 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 additional analysis demands. In this paper, we present a survey of existing lossy compressors. Then 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 such as entropy, distribution, power spectrum, principal component analysis, and autocorrelation) 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, 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 mode or online mode. The offline mode allows users to do 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 codesign 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 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 developing this document.

## 2.3 Hardware technologies

Hardware technologies are a secondary primary source of the codesign problems that Cheetah is designed to address. We have not as yet consulted with hardware projects.

# 3 Use cases

---

Use cases capture the user stories that motivate the work. As mentioned previously, Z-checker provides two execution modes based on users' diverse demands of compression assessment. It is a flexible framework being able to be added by more compressors or analysis function libraries. We first introduce the use cases related to execution modes, and then describe the use case of integrating lossy compressors into Z-checker, and finally summarized the ECP applications we 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 users can perform the analysis by different commands, such as *analyzeDataProperty*, *compareDataSets*, *runOfflineCase*, *generateGNUPlot*, and *generateReport*.

- The command, *analyzeDataProperty*, can output the property information of the target data, such as value range, auto correlation, entropy values, and frequency information.
- The command *compareDataSets* will receive two input files (original raw file and decompressed file) and output the data distortion information between them (such as maximum error, distribution of errors, auto correlation of errors, and peak signal-to-noise ratio).
- The command *runOfflineCase* is an integration of *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. It 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

SZ provides a set of easy-to-use APIs for other developers (such as ADIOS developers) to integrate Z-checker's assessment functions simply. 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.

In order to activate the MPI version of Z-checker, the user needs to install an mpi library such as MPICH and adopt “-enable-mpi” option during the installation of Z-checker. We provide two examples to do the parallel data analysis and parallel compression assessment respectively. The user 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 just needs to specifies the starting point and ending point of the compression and decompression respectively, by using Z-checker's interface, in order to get 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:

```

1  for (i = 0; I < ITER_TIMES; i++) {
2      localerror = doWork(nbProcs, rank, M, nbLines, g, h); //perform the simulation
3      if(i%50==0) //control the compression frequency over time steps
4      {
5          //make a name for the current target data property
6          sprintf(propName, "%s_%04d", varName, i);
7          ZC_DataProperty* dataProperty = ZC_startCmpr(propName, ZC_DOUBLE, \
8              g, 0, 0, 0, nbLines, M); //start compression
9          cmprBytes = SZ_compress(SZ_DOUBLE, g, &cmprSize, 0, 0, 0, nbLines, M);
10         //end compression
11         ZC_CompareData* compareResult = ZC_endCmpr(dataProperty, cmprSize);
12         sprintf(cmprCaseName, "%s_%s_%04d(1E-3)", varName, compressorName, i);
13         ZC_startDec(); //start decompression
14         decData = SZ_decompress(SZ_DOUBLE, cmprBytes, cmprSize, 0, 0, 0, nbLines, M);
15         ZC_endDec(compareResult, cmprCaseName, decData); //end decompression
16
17         if(rank==0)
18             ZC_writeDataProperty(dataProperty, "dataProperties");
19
20         //free data property generated at current time step
21         freeDataProperty(dataProperty);
22         //free compression assessment results at current time step
23         freeCompareResult(compareResult);
24         free(cmprBytes);
25         free(decData);
26     }
27     //Aggregate the residual for checking if the loop meets the convergence
    condition
28     if ((i%REDUCE) == 0) {
29         MPI_Allreduce(&localerror, &globalerror, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
30     }
31     if(globalerror < PRECISION) {

```

```
32     break;
33 }
34 }
```

### 3.3 The ECP applications that have been 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 ZFP of rate-distortion, rate speed, density spectrum, and derivatives (Morse segmentation of gradients) in several applications, including quantitative and visual analysis, visualization, and fluid dynamics simulation. 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

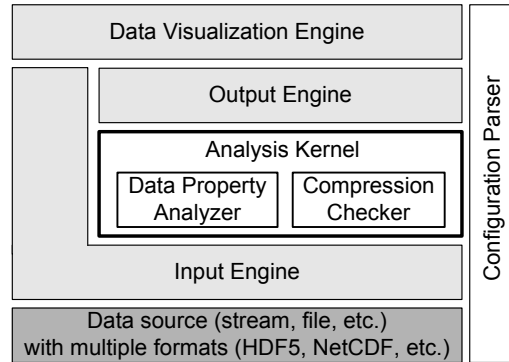
In this paper, we present Z-checker, a novel framework with three important features. (1) Z-checker can be used to explore the properties of original data sets for the purpose of data analytics or improvement of lossy compression algorithms. (2) Z-checker is integrated with a rich set of evaluation algorithms and assessment functions for selecting best-fit lossy compressors for specific data sets. (3) Z-checker 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 compute dynamically analysis 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.

**Table 1: Four Use Cases of Z-checker**

	Off-line Processing	On-line Processing
Static Display	Load data once for all, 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 for all, 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**

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 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 visualizing analysis results. The Data visualization engine also provides the interactive mode through a web browser interface (details are described later).
- *Processing module*, in the whole framework, is the core module, 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 input file path, specifying the compression command or executable, and customizing 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* (shown as the dark-gray box in the figure) is the bottom layer in the whole framework and represents the data source (such as 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 compressors:** The library that provides lossy compression ability 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 specified error-controlling.

**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 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 time 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, 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 the report.

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

### 6.3 Dependencies

Some functionalities of 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 before hand.)
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 HDF5 guide before hand.)
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=[install_dir] --enable-fftw3 --with-fftw3-prefix=DIR`; the user needs to install FFTW3 before hand.)

5. Support R language/library: the functions coded in R script can be executed in the data analysis and compression analysis. In particular, SSIM function is coded in R, so it requires 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 is an executable for users to simply test the execution of R script files. zccallr.c is a good example to show how to call 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`, `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 entrp?
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_1800_3600
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=CLDL0W_1_1800_3600:LITTLE_ENDIAN:FLOAT:1800x3600:../runOfflineCase_testdata/CLDL0W_1_1800_3600_1E-3.dat.sz
prec=1E-3
cpr_time=0.2
dec_time=0.1
cpr_data=../runOfflineCase_testdata/CLDL0W_1_1800_3600_1E-3.dat.sz
dec_data=../runOfflineCase_testdata/CLDL0W_1_1800_3600_1E-3.dat.sz.out
prec=1E-4
cpr_time=0.4
dec_time=0.2
cpr_data=../runOfflineCase_testdata/CLDL0W_1_1800_3600_1E-4.dat.sz
dec_data=../runOfflineCase_testdata/CLDL0W_1_1800_3600_1E-4.dat.sz.out
prec=1E-5
cpr_time=0.8
dec_time=0.4
cpr_data=../runOfflineCase_testdata/CLDL0W_1_1800_3600_1E-5.dat.sz
dec_data=../runOfflineCase_testdata/CLDL0W_1_1800_3600_1E-5.dat.sz.out
prec=1E-6
cpr_time=1.2
dec_time=0.6
cpr_data=../runOfflineCase_testdata/CLDL0W_1_1800_3600_1E-6.dat.sz
dec_data=../runOfflineCase_testdata/CLDL0W_1_1800_3600_1E-6.dat.sz.out

```

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

This configuration is used by only One-command assessment of Z-checker-installer.

```

#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) sz_f(1E-6),sz_d(1E-6),zfp(1E-6)"

#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 a very easy-to-use way for users to install and use Z-checker, by leveraging Z-checker-installer. In Z-checker-installer, one user can perform only one command (i.e., `z-checker-install.sh`) to install every related depending package or libraries, such as SZ, ZFP, gnuplot, latex. After the simple installation, the user just needs to execute two commands to finish the complete assessment procedure for a given data set. (1) Executing `createZCCase.sh` can generate an individual workspace based on the user's specified test case name. (2) Executing `runZCCase.sh` will perform the compression automatically by different compressors such as SZ and ZFP, and finally generate the assessment report with narratives and figures insides. in the future, If the user wants to update his/her package based on the already installed z-checker-installer, he/she can execute one command `z-checker-update.sh` for simplicity, instead of rerunning `z-checker-install.sh`.

Both Z-checker tool and Z-checker-installer package will be put in Spack later.

## 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 June 2018 will consist of the following functionalities.

- Support both online and offline analysis
- Support reading of HDF5, NetCDF, Adios data
- 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 simply on real science applications that allow the developers to select the best-fit compressors for specific applications or improve the compression quality, particularly on new Exascale machines. The metrics will be the number of applications we can assess by Z-checker, the number of I/O libraries Z-checker supports, how much we can understand the compression quality based on Z-checker, and how successful we are at helping improve each compressor.

## 8 Work plan

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

## 9 Open questions

- How do various compressors set parameters?
- How to implement parallel version for some complicated assessment algorithms, such as FFT and auto-correlation of errors?
- How to implement parallel algorithms (online analysis) based on external calls of third-party libraries such as R language?



## 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. Isenburt, “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.