

WHICH DATA FORMAT TO STORE SCIENTIFIC DATA SHOULD I USE? A PERFORMANCE ANALYSIS

Daniel GECÁŠEK, Michal SOLANIK, Ján GENČI

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, Tel. +421 55 602 3175,
E-mails: daniel.gecasek@tuke.sk, michal.solanik@tuke.sk, jan.genci@tuke.sk

ABSTRACT

A lot of scientific work is dedicated to the analysis of data. Most of the analyzed data, like data from space missions, are structured. The choice of data format can affect various characteristics - read/write speed of standard files, read/write speed of small files and read/write speed of compressed data formats. In this paper, we analyze binary data formats, proposed types of the tests and testing methods, and compare their performance with human-readable text format. We also discuss compressed and uncompressed modes available for data formats like HDF5 and netCDF. When disregarding precision, the best data format from the size perspective is lossy HDF5 without compression. Lossless HDF5 without compression show the best speed performance. Lossy HDF5 without compression is the best balance between size reduction and speed. However, for specific criteria and types of files, there might be better candidates as detailed in the conclusion.

Keywords: scientific data; data storage; compression; performance

1. INTRODUCTION

Selection of data format is important for storing and organizing data. Choice of the data format affects read/write performance and the size of output files. In [1] authors recommend usage of CSV (comma-separated values) to store tabular data with minimal metadata.

Scientific data are highly structured and may require support for storing such data. Multiple binary formats, like CDF [2], netCDF [3], and HDF5 [4], were developed to support various requirements on API (application programming interface), learning curve, and portability.

Binary and text data formats provide various read/write performances. In this article, we will analyze binary data formats, prepare methodology for testing characteristic properties of data formats, and compare obtained results. Performance results from this article are library and hardware-specific. For this reason, we performed the tests not only on the SSD (solid state drive) drive but also on a RAM (random access memory) disk. Still, the size on a disk should be applicable for any library in any programming language that implements an interface to the specific data format described here.

The data examined in this article are part of the COR (Cut-Off Rigidity) system which is briefly presented in [5]. In this article we will be concerned with two types of files:

- Normal files - these files contain a header and column data and their size is in order of kilobytes. We provide more detailed description in 4.1.
- Small files - these files contain only up to three floating point numbers each and their size is in order of bytes. We provide more detailed description in 4.2.

In the article [5], an example application of the data stored in small files is shown where the history of cut-off rigidity and cosmic ray intensity values for chosen places are examined in the context of transparency of the Earth's magnetosphere.

In the context of the COR system, normal files represent the output from cosmic ray trajectory simulation and small files are derived products from the simulated files. In the context of the article, we can look at normal files as a common example of columnar scientific data with reasonable access time performance and small files as a poorly optimal (from the point of view of access time performance) way of storing data.

In this article, we discuss performance benefits of usage of binary data formats, size and access time with/without compression. At the same time we analyse benefits and performance differences between used binary data formats with small and normal files.

2. RELATED WORK

In paper [6], the author analyzed multiple data formats and libraries on a sample from LHC (Large Hadron Collider). Metrics evaluated were: resilience against data corruption and performance of data encoding and compression. Findings relevant to this paper suggest that the ROOT format is the most performant for HPC (high-performance computing) data. Our paper focuses only on performance tradeoffs and the only examined format shared between the papers is HDF5. Moreover, the examined data has a different structure and lossy compression techniques were not considered. In [7] the authors compare the performance of various I/O (input/output) techniques applicable to netCDF format on Cray supercomputers.

In manuscript [8], the authors used gzip compression and base64 encoding to compress ASCII (American Standard Code for Information Interchange) encoded floating point coordinates of a 3D model. They achieved a good compression ratio while still preserving the text nature of the VRML (Virtual Reality Modeling Language) data format. In [9] the efficiency of various lossy compression techniques and their impact on the validity of results is examined. In the manuscript [10], the effectiveness of lossy and lossless compression on large-scale data from LHC was in-

vestigated. A new lossless compression technique was devised and performance-tested in [11].

3. POPULAR DATA FORMATS USED IN PHYSICS

In this section, we present existing data formats that are used to hold data for various domains but are also used for physics data, especially in data from space missions. Binary formats require special libraries or software to be readable.

Specifically, we will introduce CDF (Common Data Format), NetCDF (Network Common Data Form), and HDF5 (Hierarchical Data Format 5) data formats.

3.1. CDF

Common Data Format [2] is a binary data format that provides storage for scalar and multidimensional data. It is one of the most commonly used data format in the scientific field. Data in this file format are accessed via API. The standard distributed package via package managing tools like APT offers several tools that allow the user to read data contained in CDF format as they are in a non-human readable form.

3.2. netCDF

Network Common Data Form [3] is a binary data format based on CDF format. netCDF is oriented toward storing arrays of values of the same type. Storage of scalar values is possible. It can be done via initialization of the size of the array dimension to zero. Similar to the case of CDF, data are accessed via API. Also, the standard package offers tools that can convert netCDF files to human-readable format.

netCDF [3] supports compression and offers several options on how we can improve the output size of netCDF file. Keyword `least_significant_digit` can be used for specifying the smallest reliable value of real numbers. Keyword `significant_digits` allows the user to specify the absolute number of significant digits independent of the magnitude of the variable.

3.3. HDF5

Hierarchical Data Format [4] is a binary data format designed to store scientific data. HDF5 is self-describing, it can store metadata in global or variable attributes. It supports the compression of data. The most significant difference between HDF5 and CDF is data grouping. HDF5 allows grouping objects into a group. API of HDF5 is identical to API of netCDF data format and it is possible to use HDF5 API on netCDF file and vice versa.

HDF5 also [4] supports compression. It allows users to define specific filters, like LZ4, LZ4HC, etc., that can decrease the size of output data. One of the available filters, the scale-offset filter allows definition of the precision of float values.

4. DATA SPECIFICATION

In this article, we compare existing data formats suitable to hold the output of physical simulation of trajectories of cosmic ray particles in the Earth's magnetosphere. In this section, we present the description of the data used for performance analysis.

4.1. Description of testing data for normal files

Each file holds two types of data - metadata and array data. Metadata can be found in the header and footer parts of the file. The rest of the file contains column data. The structure of column data that needs to be stored is shown in Table 1.

Together the header is 432 characters long and the footer is 79 characters long where useful information takes at most 76 characters in the header and 23 characters in the footer together with one space as the separator between values. The sample file used in experiments contains 9288 lines of column data.

Value in column 0 shown in Table 1 is printed to 6 decimal places however, it is rounded to two decimals and the rest are zeroes.

Table 1 Structure of main file column data

Column index	Decimals	Upper bound length
0	6 (2)	8
1	10	12
2	6	9
3	3	7
4	3	7
5	3	7
6	3	7
7	6	8
8	2	8

All lines in text data format have the same length of 107 characters for column data to improve readability. Easy reduction of file size can consist of removal of alignment spaces which, in the worst-case scenario would mean a reduction to 88 characters without rounding of the first line or 84 characters with rounding to two decimal places. Compared to $4 \times 9 = 36$ bytes in the case of nine single-precision floating-point numbers or $8 \times 9 = 72$ bytes in the case of double precision. This by itself brings a reduction of size to 40.61% of the original in the case of optimized ASCII form and 33.64% reduction of the original size in the case of currently employed ASCII.

4.2. Description of testing data for small files

Small files contain only a line consisting of either 3 floating-point numbers with values of cut-off rigidities or one floating-point value, containing proton intensity as shown in Table 2.

Table 2 Structure of testing data for small files

Quantity name	Filename	Decimals	Length [b]
Intensity	intensity	6	10-11
Lower cut-off	cutoff	2	3-4
Upper cut-off		2	3-4
Effective cut-off		2	3-4

As is obvious from the table, the full sizes of small files vary. A file that contains intensity can be either 10 (low and middle geomagnetic latitudes) or 11 (high geomagnetic latitudes) bytes long. A file that contains cut-off rigidities ranges from 11 (all values are < 10) to 14 (all values are > 10) bytes.

For testing, we generated 60000 pairs of small files (one intensity and one cut-off file per pair).

5. APPLICATION IN SCIENTIFIC FIELD

Usually, scientific missions produce a large amount of data. Most of the data are available via public APIs dedicated to these missions. Some data are available in popular data formats. We will discuss several scientific missions and analyze available files for non-human readable data formats.

Voyager [12] is a space mission that involves the launch of two spacecraft for exploring Jovian, Saturnian systems, and also interplanetary space. Mission has published a large amount of data since 1977 and consists of multiple human and also non-human readable data formats. Following non-human readable data, following data formats are used: CDF, DAT, and vmsbin for storing images.

Parker Solar Probe [13] is a space mission dedicated to the measurement and analysis of data from the Sun's atmosphere. Launched spacecraft has multiple instruments, from which are data collected and published. Following non-human readable data formats are used: CDF and spice kernel data format.

OCO-2 (Orbiting Carbon Observatory 2) [14] is satellite used to monitor various environmental indicators. Following non-human readable data formats are used: netCDF, HDF5.

JPSS-1 (Joint Polar Satellite System 1) (later known as NOAA-20) [15] is the name of the satellite that is used to monitor the Earth's atmosphere. Its purpose is to provide various data needed for numerical models for weather prediction and climate change. Output file data are in HDF5 file format.

For a better understanding of the overall usage of these data formats, we used the NASA Abstract search engine to determine trends in using data formats in the field of physics. In Table 3 are shown results from search engine for a found number of results for each searched data format. The popularity of CDF can be explained by its usage in big scientific institutions as a standard data format.

Table 3 Found number of results for data formats

Searched text	Number of results
CDF	58 628
NetCDF	2 802
HDF5	2 301

We can observe move to different data formats in newer NASA space missions. The Traditional CDF data format is still used, but some newer NASA missions started to use HDF5 for storing complex data.

6. METHODOLOGY

We decided to test performance metrics of data formats with compressed and uncompressed versions of CDF, HDF5, netCDF¹ and pure text data formats. In the article, we refer to the least_significant_digit keyword of netCDF and scale-offset filter of HDF5 as lossy compression.

For compression of CDF, and netCDF we used gzip and zlib respectively. For compression of HDF5 files, we tried both gzip and lzf compressions available out of the box. For compression of text files, we used the zip command on files recursively (-r) with suppressed output (-q) and the highest possible compression ratio (-9). The following metrics are tested:

- output size of tested data in uncompressed / compressed / lossy uncompressed / lossy compressed state,
- read / write speed in uncompressed state,
- read / write speed in compressed state,
- read / write speed in lossy uncompressed / compressed state.

Results presented in this work are obtained from 500 runs of the proposed tests to decrease the impact of any statistical deviations to a minimum. Besides statistical deviations, we will present results from SSD and RAM disk² to minimize the impact on disk performance. The measured parameter of each set of test runs was the median value. We focus on runtime - execution time of proposed test and size of the output files without compression and compression.

Input data is generated for small file tests or loaded from a sample file in the case of normal files. The range of values generated is limited to expected values of intensities and cut-off rigidities, however, the distribution is not controlled. For data generation, a pseudo-random number generator from Python standard library³ is used with a seed constant for all tests. Data for the normal test is taken from a chosen simulation file. Parameter for lossy formats that determines precision is set to two decimals in the case of normal files and number of decimals in Table 2 in the case

¹For testing, we used the following libraries: CDFlib, netCDF4, and h5py

²tmpfs

³<https://docs.python.org/3/library/random.html>

of small files. The proposed order of testing steps is presented in Figure 1.

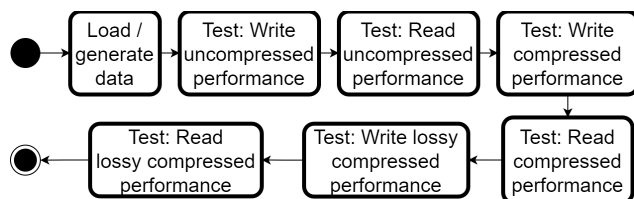


Fig. 1 Order of test cases run on both SSD and RAM disk

After each testing step, the operating system cache is deleted. All testing steps repeated in all test iterations can be seen in Figure 2. Time measured in each iteration is stored until all iterations finish and then the median time is evaluated. Size measurements of output files are integrated with reading performance testing steps and happen only once after testing.

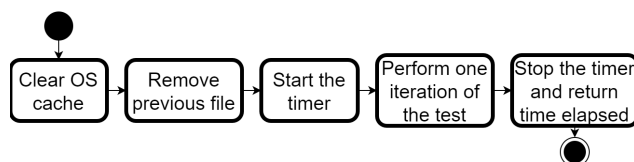


Fig. 2 Flow of each test case displayed in Figure 1. Notice that the timer is started after the cache is cleared and previous files are deleted. This ensures that only useful performance is timed

When converting the normal file to binary format, we stored the values of each column in one array. In the case of small files, we stored all quantities from Table 2 into a file with one array per quantity.

For testing, we used a computer with an i7-9750H CPU (Central processing unit), 16 gigabytes 2666MHz of RAM, and Intel 660p 512 gigabyte SSD as a test machine. We decided to run separate tests for normal and small files as performance with small files is different compared to normal files.

7. RESULTS

In the following section, we discuss results from tests devised in section 6.

7.1. Size comparison

In the default case where we measured the size of the normal file, shown in Table 4, size reduction was caused just by conversion to binary data format. That is expected because each character in ASCII representation is one byte which means that the size a value occupies scales linearly with the number of digits. In the case of floating-point numbers, the existence of a decimal separator increases the size of each number by one byte. For the case of binary numbers encoded by IEEE 754 standard, each number occupies the same size as the other with no regard to size and precision. That means that in extreme cases information is lost.

Usage of binary formats shows the reduction in an interval from 30.69% to 31.01% of the original file size. The difference between data formats themselves is minimal.

Table 4 Size of normal files for data formats without compression (in kilobytes)

	text	netCDF	HDF5	CDF
Lossless	983.77	681.83	678.50	678.32
Lossy	-	682.66	129.98	-

Reduction of floating-point precision is offered by netCDF and HDF5. In the case of reduction of floating-point precision without compression we see that in the case of netCDF the size even increased from the case with no loss of precision. In the case of HDF5, the size reduction was enormous being 86.79%.

Considering compression as seen in Table 5 without loss of precision, results of binary formats are mostly similar with an outlier being CDF, which reduced the file size by 79.72%. Reduction in the case of other formats was in an interval from 59.53% (HDF5 gzip) to 66.06% (HDF5 lzf). Interestingly, the best compression was achieved when the text format was compressed. This was probably in part caused by the usage of double-precision to store data with binary formats.

When considering lossy compression (only offered by netCDF and HDF5) the biggest reduction was with netCDF being 91.29%. HDF5's gzip and lzf compression methods achieved similar results in a reduction, being 88.18% and 87.86% respectively. The biggest size reduction was achieved when using lossy compression with netCDF.

Table 5 Size of normal files for data formats with compression (in kilobytes)

	text	netCDF	HDF5		CDF
			gzip	lzf	
Lossless	180.13	336.73	333.93	398.15	199.50
Lossy	-	85.69	116.28	119.44	-

In the case where we measured the size of small files shown in Table 6, size increase was seen in conversion to binary data format. In this case, rounding to two decimals and small values of rigidity numbers (< 100) were shown as a much more efficient storing method than binary doubles. The increase of size compared to text representation was 18% when using binary format in lossless form without compression.

Considering lossy representation, the size in the case of netCDF again increased by about 18%. In the case of HDF5, a significant reduction to 27% of the original size was achieved.

Table 6 Size of small files for data formats without compression (in kilobytes)

	text	netCDF	HDF5	CDF
Lossless	1625.71	1926.68	1924.48	1922.84
Lossy	-	1927.58	441.984	-

For compression formats seen in Table 7, the text compression did not decrease the size of the file but, on the contrary, the size has increased to 1419% of the original size.

To investigate we run the zip command on a sample of pair of 60 small files. The output of zip command showed that zip used its compression method on 32 files out of 120. After examination of the zipped file, it is obvious that only some compression was done. The zip contains filenames and their content on the same line and additional zeros that have no obvious purpose to us between the files.

Usage of lossless compressed binary formats showed the change of size to 93%, 102%, 109%, and 49% of the text file size for netCDF, HDF5 with gzip, HDF5 with lzf, and CDF respectively.

When considering lossy compression, the sizes of all formats showed a significant decrease from the text file size to 30%, 27%, and 27% for netCDF, HDF5 with gzip, and HDF5 with lzf respectively.

Table 7 Size of small files for data formats with compression (in kilobytes)

	text	netCDF	HDF5		CDF
			gzip	lzf	
Lossless	23080.59	1518.63	1654.44	1779.79	798.22
Lossy	-	480.67	442.61	442.12	-

The differences in size reductions of one normal file and small files comes from number of formatting white-space characters and number of characters a number is composed. The general trend is that the more white space characters and characters are in the ASCII file, larger the size reduction from usage of binary data formats.

7.2. Runtime comparison

In Tables 8 to 15, we show the raw results of the experiments. Generally, the performance of binary data file formats is better than the performance of text data formats. Text data formats require an additional step of formatting data before the write and also parsing data during the reading of files.

Tables 8 to 11 contain the runtime performance of normal files for chosen data formats. In these tests, no significant differences were found between tests on SSD and RAM, except runtime performance on RAM was a bit better.

Tables from 12 to 15 contain the runtime performance of normal files for chosen data formats. During these tests, we generated 60000 pairs of small files. Runtime performance

between text data format and binary data formats has increased even more. The most possible explanation for this is that each small file requires open, write/read, and close system calls and also formatting/parsing data depending on the type of operation. Tested binary data formats offer the advantage of additional metadata and data structure that the user can use to make them more efficient during the reading operation. The data structure can also affect the efficiency of the writing operation.

Table 8 Median runtime performance of normal files for chosen data formats without compression using SSD (in ms)

	text	netCDF	HDF5	CDF
Write	35.32	8.09	5.51	3.46
Read	24.44	8.84	5.20	5.25
Write lossy	-	8.85	12.63	-
Read lossy	-	9.00	5.56	-

Table 9 Median runtime performance of normal files for chosen data formats without compression using RAM (in ms)

	text	netCDF	HDF5	CDF
Write	34.83	7.38	5.22	3.30
Read	19.12	4.18	2.25	2.54
Write lossy	-	8.02	12.38	-
Read lossy	-	4.28	4.22	-

Table 10 Median runtime performance of normal files for chosen data formats with compression using SSD (in ms)

	text	netCDF	HDF5		CDF
			gzip	lzf	
Write	285.68	26.15	18.56	9.79	32.23
Read	46.76	9.46	7.08	5.74	5.25
Write lossy	-	23.92	16.04	13.56	-
Read lossy	-	7.38	7.08	5.84	-

Table 11 Median runtime performance of normal files for chosen data formats with compression using RAM (in ms)

	text	netCDF	HDF5		CDF
			gzip	lzf	
Write	282.63	24.42	17.40	9.19	33.8
Read	38.59	6.41	5.29	3.60	6.50
Write lossy	-	22.25	15.30	13.05	-
Read lossy	-	5.69	4.78	4.34	-

Table 12 Median runtime performance of small files for chosen data formats without compression using SSD (in ms)

	text	netCDF	HDF5	CDF
Write	3301.75	10.46	8.85	7.33
Read	15568.44	8.57	5.59	6.04
Write lossy	-	12.52	27.51	-
Read lossy	-	8.48	8.77	-

Table 13 Median runtime performance of small files for chosen data formats without compression using RAM (in ms)

	text	netCDF	HDF5	CDF
Write	1615.27	8.93	7.54	7.69
Read	1430.97	2.67	1.55	2.36
Write lossy	-	11.27	25.94	-
Read lossy	-	2.77	7.02	-

Table 14 Median runtime performance of small files for chosen data formats with compression using SSD (in ms)

	text	netCDF	HDF5		CDF
			gzip	lzf	
Write	6103.37	134.13	56.63	22.36	123.67
Read	4477.46	14.79	13.96	8.24	17.30
Write lossy	-	110.56	36.11	30.53	-
Read lossy	-	10.54	10.19	8.93	-

Table 15 Median runtime performance of small files for chosen data formats with compression using RAM (in ms)

	text	netCDF	HDF5		CDF
			gzip	lzf	
Write	3488.90	119.16	50.24	20.08	128.54
Read	2498.23	9.78	10.24	3.27	15.48
Write lossy	-	99.50	33.43	28.87	-
Read lossy	-	8.33	8.38	7.19	-

8. DISCUSSION

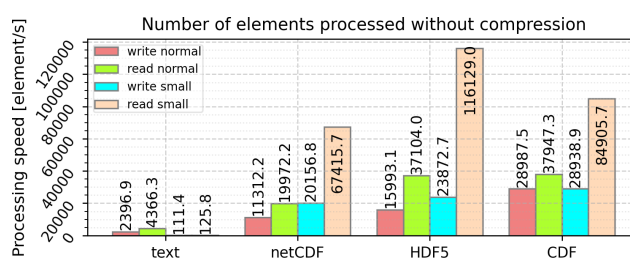
In this section, we will discuss the results with the help of figures that show differences between formats based on processing speed (number of floating-point numbers that were processed per second) and compression level per second (compression ratio divided by the runtime of examined operation in seconds). The value of compression level per second:

- shows us whether the filesize decreased or increased compared to the uncompressed text format (based on the sign of the value);
- shows which data format is the best performing when both, compression ratio, and runtime speed are important i.e. the best tradeoff of speed for size reduction (the bigger value means better ratio).

All visualized values were computed from data presented in the previous section. Times used for this analysis are those achieved from RAM disk measurements to minimize the impact of disk drive performance. Based on the results from the measurements on SSD, the same formats would work the best on SSD.

8.1. Processing speed

In Fig. 3 and Fig. 4 the processing speed of formats without compression is shown. In both cases write operation was significantly slower than the read operation. In the case of lossless storage, HDF5 was generally the fastest, followed by CDF, netCDF, and lastly text format.

**Fig. 3** Number of processed elements per second for chosen data formats without compression

We note that the text format was more performant in the case of normal files compared to small files. That is mainly caused by the minimization of the number of open() system calls described earlier. The minimization also causes improved performance of binary formats on small files. In the case of normal files, the increased performance of binary formats is most likely caused by the minimization of the number of read() calls and no need to convert data from text to IEEE 754 floating-point format. Another exceptional observation is that CDF offers faster access than HDF5 for normal files and faster write speed in the case of small files than HDF2.

In the case of lossy storage, netCDF was faster than HDF5. However, we must underline the fact that lossy netCDF storage without compression does not bring any file size decreases. This is because netCDF python library only decreases the precision of values but does not alter the way data is stored, this only helps the compression algorithm to be more efficient. When comparing speeds that netCDF attains in lossless and lossy cases, we can observe that they are essentially the same. Speed loss of HDF5 was significant but reduction of the file size was dramatic.

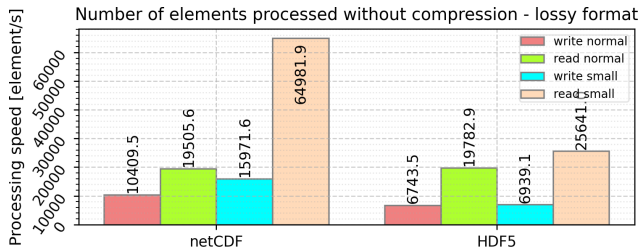


Fig. 4 Number of processed elements per second for chosen data formats without compression in lossy form

In Fig. 5 to Fig. 6 the processing speed of compressed formats is shown. In both cases write operation was significantly slower than the read operation. In the case of lossless compression, text format is the slowest in all metrics. Binary formats are mostly equally fast except HDF5 using lzf compression which is faster in the case of reading by a huge margin and a bit faster than other formats in the case of writing.

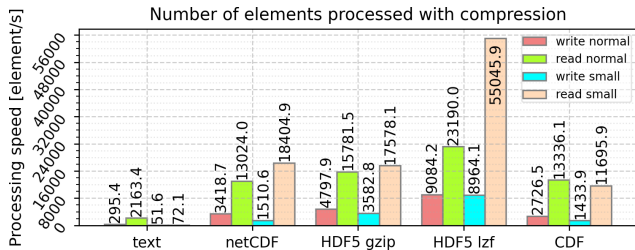


Fig. 5 Number of processed elements per second for chosen data formats with lossless compression

In the case of lossy compression, the performance of netCDF was comparable to HDF5 with gzip compression. HDF5 with lzf compression had the best performance in all categories.

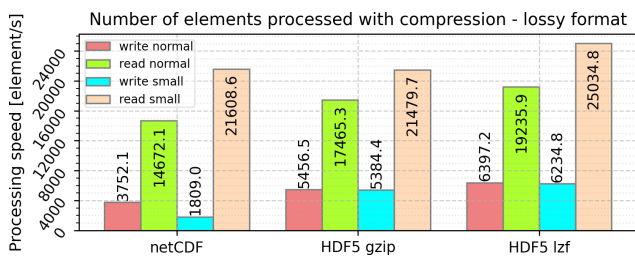


Fig. 6 Number of processed elements per second for chosen data formats with lossy compression

From this comparison, it is obvious that in the general case, when only speed is important, the fastest data format is HDF5 without compression. If compression is required but speed is more important than the compression ratio itself, HDF5 with lzf filter offers the best performance. If further compression is needed and precision of results is not required, lossy HDF5 with lzf compression is the fastest performing data format.

8.2. Size reduction per time

In this subsection, we examine percentual file size reduction (or increase) per unit of time (second). In our opinions, this ultimately shows which format is the best when we value both, the size reduction of data and processing time. All comparisons are made relative to the uncompressed text format. For this reason graphs for uncompressed text format are never drawn.

In Fig. 7 and Fig. 8 the file size reduction per second of uncompressed formats is shown. We can conclude that in the case of small files the file size increased in the case of all examined formats. This was caused by the usage of double-precision of floating-point values which occupy more space than values formatted as text as described in 4.2. In the case of normal files, there was a reduction of filesize because text format is less space-efficient than binary even when double precision is used. For normal files, CDF had the biggest size reduction per second, followed by HDF5 and netCDF. The reading ratio of HDF5 and CDF is comparable, however, the writing ratio of CDF was better. For small files, HDF5 increased the size per second the fastest followed by CDF and netCDF.

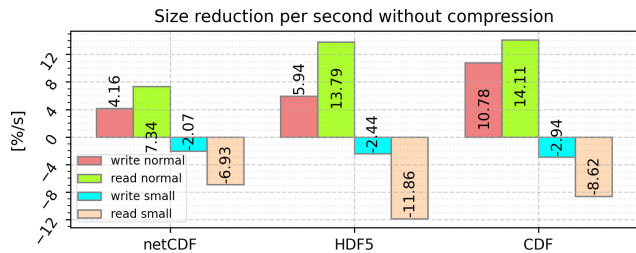


Fig. 7 Percentual size reduction (or increase) per second of chosen data formats without compression

In the case of lossy formats, netCDF shows almost no difference compared to its non-lossy version. HDF5 shows huge improvements compared to the lossless version in the case of small files. In the case of normal files, the compression per second is improved compared to the non-lossy version.

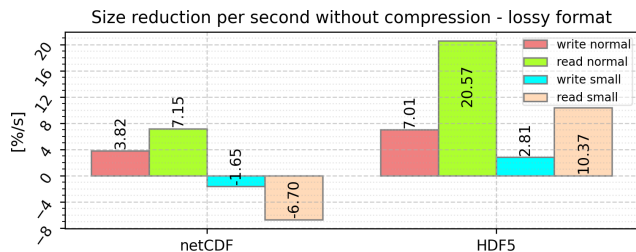


Fig. 8 Percentual size reduction (or increase) per second of chosen data formats without compression in lossy form

In Fig. 9 and Fig. 10 the file size reduction per second of uncompressed formats is shown. In the case of lossless compression of normal files, we can observe that HDF5 with lzf compression had the best ratio per second followed by HDF5 with gzip CDF, netCDF, and finally compressed text. In the case of small files, we see that both HDF5 and

text format increased the filesize, netCDF decreased the file size with a small decrease per second and CDF decreased the filesize with the biggest compression per second.

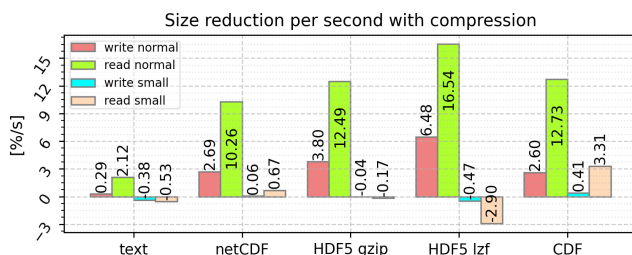


Fig. 9 Percentual size reduction (or increase) per second of chosen data formats with lossless compression

In the case of lossy compression, HDF5 using lzf had the best compression ratio per second, followed by HDF5 using gzip and lastly netCDF.

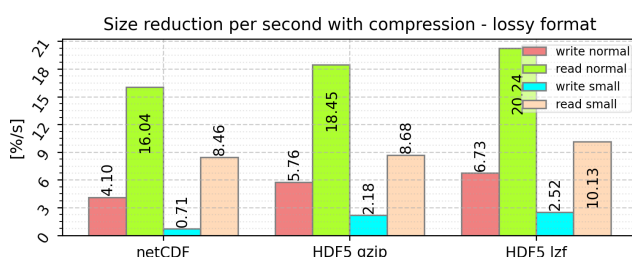


Fig. 10 Percentual size reduction (or increase) per second of chosen data formats with lossy compression

When considering both, size reduction and speed, the best format to store normal files that preserve input precision is either uncompressed CDF or HDF5 with lzf. HDF5 with lzf offers a better compression ratio to read speed while CDF is better when data is mostly read. In the case of small files, the best data format to use is compressed CDF which achieved file size reduction with a better ratio than netCDF which was the only other format where file size reduction was achieved on small files.

When precision is not that important, lossy HDF5 without compression offers the best tradeoff of speed for size reduction for both normal and small files.

9. CONCLUSION

In this article, we compared text and binary data formats in terms of their runtime and compression capabilities. The data we chose to compare are cosmic ray trajectory simulation outputs and products derived from them. The trajectory data were chosen from a real simulation while values of derived products were generated randomly to fit in their respective value ranges. In the case of binary formats, we chose to store the data in double-precision format to avoid loss of precision.

When measuring runtime performance, we decided to measure performance on SSD but a RAM disk to see the impact of SSD on performance. That showed that, predictably, using SSD was slower than RAM except for the case of compressed CDF. The order of formats from fastest

to slowest was mostly preserved regardless of the storage medium used.

From the perspective of the size of the file, the best lossless format for normal files was compressed text files and the best lossy format was compressed netCDF. In the case of small files, file sizes with lossless binary formats mostly increased except for netCDF and CDF where CDF achieved much better results. In the case of lossy formats, the best one was uncompressed HDF5.

When analyzing processing speed, we can observe that reading is faster than writing in the case of all data formats and the processing speed of small files is better than normal files. Considering processing speed, HDF5 without compression was the fastest and it was also the fastest in most other cases except for the uncompressed lossy case where netCDF was faster (but increased file size).

When valuing both compression and speed, we can ignore data formats that increase the filesize. For lossless data formats, the best ratio of compression to time was achieved with HDF5 using lzf compression or uncompressed CDF for normal files and CDF for small files. The absolute best format when considering both size reduction and read/write speed and disregarding precision is lossy HDF5 without compression. It offers the best performance for both normal and small files.

The main takeaway from this article is that using any binary format with appropriate structure will increase read and write access speed to data while also providing smaller file sizes in most cases. In the case of small files, the improvement in read and write access speed is enormous when using binary data formats.

REFERENCES

- [1] Van den EYNDEN, V. – CORTI, L. – WOOLLARD, M. – BISHOP, L. – HORTON, L.: *Managing and sharing data; a best practice guide for researchers*, UK Data Archive, No. () 2011
- [2] Space Physics Data Facility NASA / Goddard Space Flight Center: *CDF User's Guide*, No. 3.8.1 (2021)
- [3] REW, R. – DAVIS, G.: *NetCDF: an interface for scientific data access*, IEEE computer graphics and applications **10**, No. 4 (1990) 76–82
- [4] KOZIOL, Q. – ROBINSON, D. et al.: *Hdf5*, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), No. (2018)
- [5] GECÁŠEK, D. – BOBIK, P. – GENCI, J.: *Transparency of magnetosphere for cosmic rays in last two millennia*, Proceedings of 37th International Cosmic Ray Conference — PoS(ICRC2021) **395**, No. - (2021) 1335
- [6] BLOMER, J.: *A quantitative review of data formats for HEP analyses*, Journal of Physics: Conference Series **1085**, No. 3 (2018) 032020
- [7] BALLE, T. – JOHNSEN, P.: *Improving i/o performance of the weather research and forecast (WRF) model*, Cray User Group 2016, No. (2016) 123
- [8] ISENBURG, M. – SNOEYINK, J.: *Binary compression rates for ASCII formats*, Proceedings of the

- eight international conference on 3D Web technology, No. (2003) 173
- [9] LANEY, D. – LANGER, S. – WEBER, Ch. – LINDSTROM, P. – WEGENER, A.: *Assessing the effects of data compression in simulations using physically motivated metrics*, Scientific Programming **22**, No. 2 (2014) 141
- [10] AUBERT, P. – VUILLAUME, T. – MAURIN, G. – JACQUEMIER, J. – LAMANNA, G. – EMAD, N.: *Polynomial data compression for large-scale physics experiments*, Computing and Software for Big Science **2**, No. 1 (2018) 1–9
- [11] BERGER, J. – FRANKENFELD, U. – LINDENSTRUTH, V. – PLAMPER, P. – RÖHRICH, D. – SCHÄFER, E. – SCHULZ, M. W. – STEINBECK, T. M. – STOCK, R. – SULIMMA, K. et al.: *TPC data compression*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **489**, No. 1-3 (2002) 406–421
- [12] KOHLHASE, C. E. – Penzo, P. A.: *Voyager mission description*, Space science reviews **21**, No. 2 (1977) 77–101
- [13] WHITTLESEY, P. L. et al.: *The solar probe analyzers—electrons on the parker solar probe.*, The Astrophysical Journal Supplement Series **246**, No. 2 (2020) 74
- [14] BASILIO, R. R. – POLLOCK, H. R. – HUNYADILAY, S. L.: *OCO-2 (Orbiting Carbon Observatory-2) mission operations planning and initial operations experiences.*, Sensors, Systems, and Next-Generation Satellites XVIII. **9241**, No. (2014) 31–42
- [15] ZHOU, L. – MURTY D. – XINGPIN, L.: *An overview of the Joint Polar Satellite System (JPSS) science data product calibration and validation.*, Remote Sensing **8**, No. 2 (2016) 139

Received May 1, 2022, accepted June 13, 2022

BIOGRAPHIES

Ján Genčí assoc. Prof. in Applied Informatics. He works at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice (TUKE), since 1984. His research interests are tightly connected with the subjects he teaches – data processing.

Daniel Gecášek was born on 3.4.1995. In 2019 he completed his master's degree at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He defended his PhD thesis in the field of computer science in 2022; his thesis title was "Methods and algorithms of automated processing of data and models of cosmic physics". His scientific research is focusing on the automation and post-processing of particle physics simulations.

Michal Solanik is a PhD student of computer science at TUKE. He completed his master's degree with thesis "Parallelization of two-dimensional models of cosmic rays modulation in heliosphere". In the process of his PhD studies at TUKE, he continues with parallelization, acceleration and automation of parallelization for models of cosmic rays modulations in heliosphere.