Chimp: Efficient Lossless Floating Point Compression for Time Series Databases

Athens University of Economics and Business

Panagiotis Liakos - Katia Papakonstantinopoulou - Yannis Kotidis

Athens University of Economics and Business

ΕΛΙΔΕΚ Ελληνικό Ίδρυμα Έρευνας & Καινοτομίας

What's this about?

We propose a novel streaming compression algorithm for floating point values. We focus on **Time series Management Systems** and examine the efficiency of our techniques by extending `InfluxDB`. **Our contributions:**

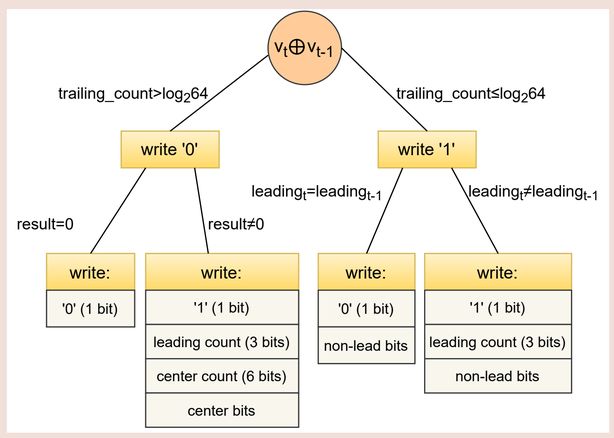
i) we perform a **survey** of compression algorithms.

ii) we uncover **important properties** of floating point time series.

iii) we greatly reduce the space requirements for compressing floating point data, requiring on average about **half of the space** of similar state-of-the-art approaches.

iv) we significantly outperform general purpose algorithms, and also surpass the performance of streaming techniques in **terms of speed**, offering an overall **greatly improved trade-off** between space and speed, which clearly stands out compared to earlier efforts, streaming or not.

Improving Gorilla[1] with Chimp



Chimp **improves the state-of-the-art** compression algorithm by using:

- trailing zeros *only* when their number is larger than $\log_2 64$.
- the number of previous leading zeros *only* when it is the same with the current one.
- just 3 bits to specify the number of leading zeros.

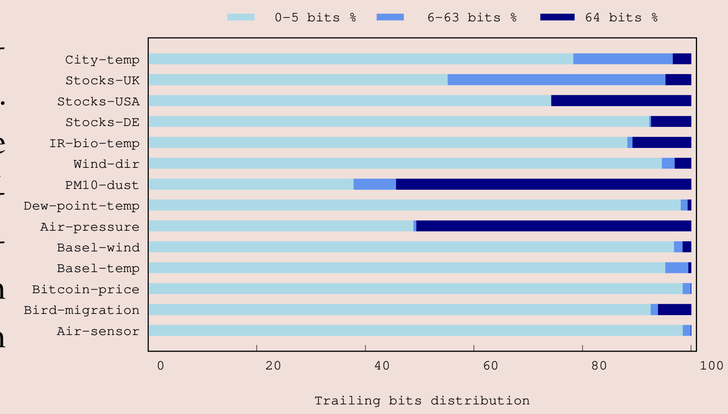
[1] Pelkonen et al. "Gorilla: A fast, scalable, in-memory time series database." Proc. of the VLDB Endowment 8.12 (2015).

XOR-Based (⊕) Floating-Point Compression

An effective compression technique is to perform a **bitwise XOR** operation between the current value and the previous value. The resulting set of bits is likely to contain a lot of **leading zeros**, as the sign and exponent are often identical for neighboring data points. XOR-based compression also attempts to exploit **trailing zeros**:

```
0.2: 00111111 11001001 10011001 10011001 10011001 10011001 10011001 10011010
0.4: 00111111 11011001 10011001 10011001 10011001 10011001 10011001 10011010
0.8: 00111111 11101001 10011001 10011001 10011001 10011001 10011001 10011010
2.2: 01000000 00000001 10011001 10011001 10011001 10011001 10011001 10011010
```

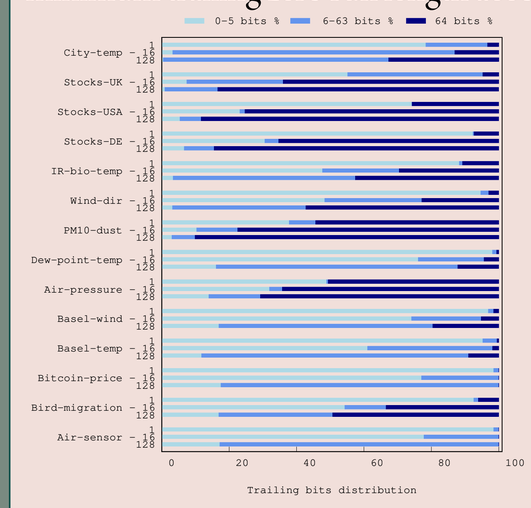
However, long runs of trailing zeros are **not very often**. Thus, we can improve the space-efficiency of state-of-the-art approaches by utilizing trailing zeros **only** when their number is large enough to provide savings.

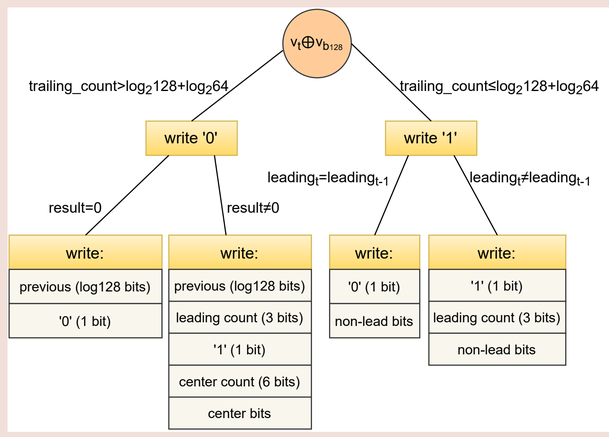


Exploiting more previous values with Chimp$_{128}$

If we use **the best of the previous 128 values**, with regard to the largest run length of trailing zeros, we can come up with much longer runs.

Chimp$_{128}$ uses $log_2 128$ **index bits** to specify the previous value used, and adjusts the minimum trailing zero run length accordingly.





Experimental Evaluation

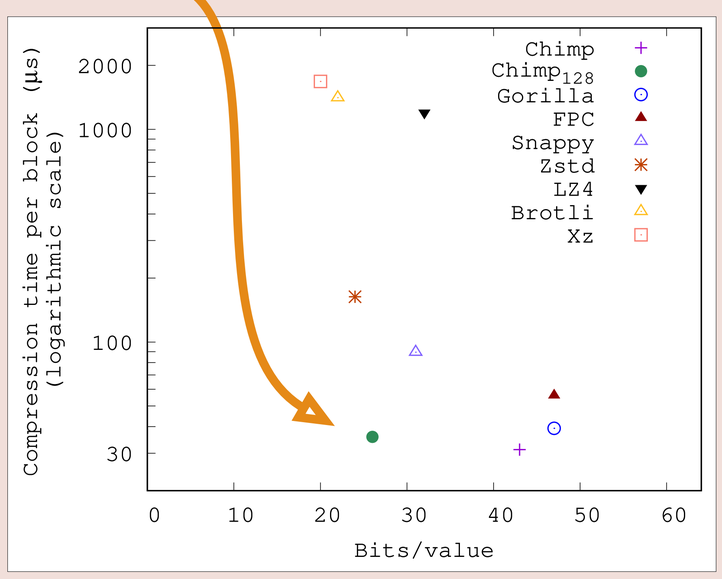
Our dataset consists of **14 time series** and **5 non time series** datasets.

Execution Environment: A PC running Xubuntu 20.04 with an Intel® CoreTM i5-4590, with a CPU frequency of 3.30GHz and a 6MB L3 cache, and a total of 16GB DDR3 1600MHz RAM.

Our evaluation answers the following questions:

- What are the **space requirements** of Chimp$_{128}$ compared to earlier approaches?
- How does the **trade-off** of Chimp$_{128}$ between compression ratio and compression time compare to the state-of-the-art?

	Dataset	General Purpose Compression					Streaming Compression			
		Xz	Brotli	LZ4	Zstd	Snappy	FPC	Gorilla	Chimp	Chimp$_{128}$
Time series	City-temp	14,04	15,31	27,64	17,90	24,30	55,16	58,72	46,21	22,92
	Stocks-UK	7,61	8,54	19,84	10,32	15,80	46,15	33,45	31,27	16,70
	Stocks-USA	7,19	8,11	18,16	9,92	14,68	36,02	36,43	34,67	12,06
	Stocks-DE	8,80	9,96	20,63	12,06	16,83	44,54	45,63	42,88	13,46
	IR-bio-temp	13,82	16,05	29,13	20,19	25,58	48,52	50,33	46,39	18,94
	Wind-dir	12,66	14,98	26,95	17,88	22,02	58,12	59,62	54,31	19,80
	PM10-dust	6,55	7,21	15,03	8,50	12,52	27,79	26,91	24,40	13,64
	Dew-point-temp	20,92	25,16	38,34	29,65	38,60	53,63	54,42	51,57	32,49
	Air-pressure	14,35	14,96	21,66	17,23	21,61	24,07	23,96	22,92	19,23
	Basel-wind	36,77	38,93	44,20	38,96	47,57	58,75	63,63	56,09	45,65
	Basel-temp	22,22	25,10	34,67	26,06	34,51	57,58	60,19	54,10	30,12
	Bitcoin-price	40,29	46,46	55,20	47,64	63,19	52,22	52,50	49,68	47,17
	Bird-migration	24,97	27,11	35,50	29,12	34,00	48,14	50,24	45,92	28,37
	Air-sensor	50,16	54,22	64,32	58,53	64,10	52,56	52,98	49,54	49,56
	Time series average	20,03	22,29	32,23	24,57	31,09	47,38	47,79	43,57	26,44
Non time series	Food-prices	16,32	17,87	27,65	19,96	26,28	43,53	37,94	27,92	24,59
	POI-lat	39,30	41,94	50,19	43,08	52,81	60,65	65,95	57,80	47,71
	POI-lon	43,97	46,46	54,16	47,67	57,24	63,77	66,07	62,71	54,55
	Blockchain-tr	45,00	47,82	54,90	48,50	59,11	60,10	62,83	58,25	53,16
	SD-bench	8,12	8,98	19,05	10,65	15,70	37,74	40,25	35,10	17,00
	Non time series average	30,54	32,61	41,19	33,97	42,23	53,16	54,61	48,36	39,40



these findings establish our structures as the *undisputed preferable* option for time series databases!

Future Directions - Contact

We plan to extend our work by investigating **lossy** streaming compression techniques that allow for recreating the time series within a very small error bound.

This work has received funding from HFRI and GSRT, under grant agreement No 779.

Contact info: http://isdb.cs.aueb.gr/delorean/