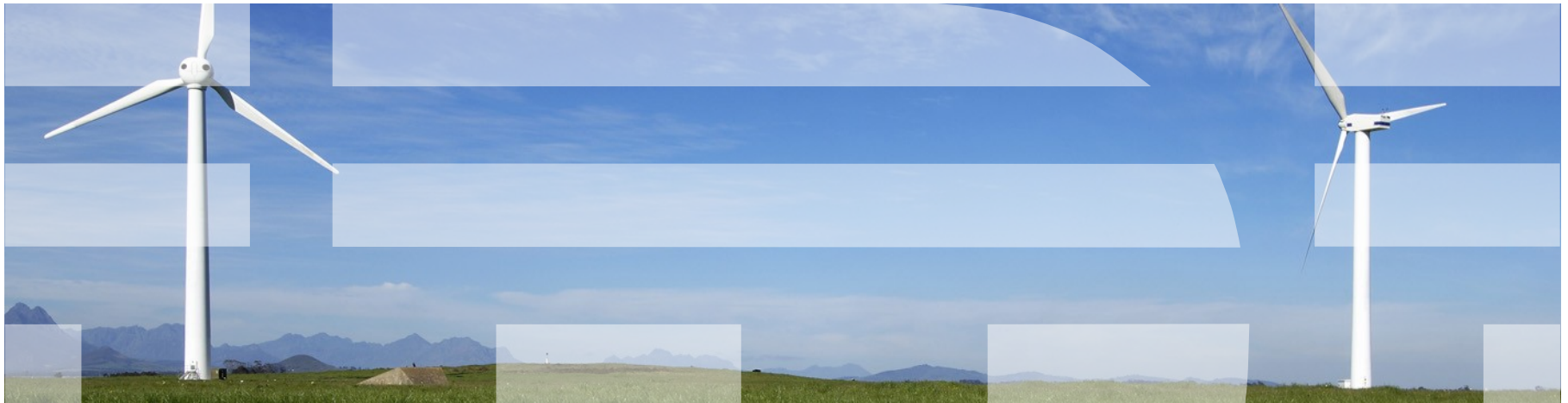
To Zip or not to Zip – Effective Resource Usage for Real-Time Compression

Danny Harnik, Oded Margalit, Ronen Kat, Dmitry Sotnikov, Avishay Traeger
IBM Research - Haifa



Our scope – Real-Time Compression

- Compression for **primary data** in **enterprise** storage systems
- Many benefits to compression. Reduces:
  - Costs
  - Rack space
  - Cooling
  - Can delay need for additional purchases to existing systems

- **The challenge:** Add “seamless” compression to a storage system with little effect on performance

A bit about compression techniques:

- We focused on Zlib – a popular compression engine (zip). Combines :
  - **Lempel Ziv [LZ77] compression** – pointers instead of repetitions
  - **Huffman Encoding** – use shorter encoding to popular characters

Estimating compression ratios – some motivation

- To Zip or not to Zip?
  - Compression does not come for free
  - Incurs overheads, sometimes significant
  - Not always worth the effort – depends on the actual **data**
  - **Goal:** Avoid compressing "incompressible" data

Especially In a **storage system** with a high disk to server ratio

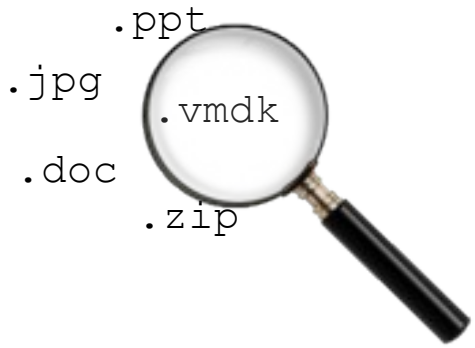
Other potential benefits:

- **Evaluation and sizing**
  - Compression ratio → number of disks → money!
  - **Evaluation:** should I invest in a storage system with compression
  - **Sizing:** How many disks should I buy

Existing solutions

Rules of thumb

- Deduce from past experience with similar applications
- By file extension
  - Not always accurate
  - Not always available



Look at the actual data

- Scan and compress everything
  - Takes too long
- Look at a prefix (of a file/chunk) and deduce about the rest
  - No guarantees on the outcome

- No established method for estimating compression ratio other than compressing
- Raskhodnikova et al. [RRRS 2007] – Cannot accurately estimate LZ77 compression without reading essentially all of the data!

Our Work – Two granularities

Macro scale

- Large volumes
  - GigaBytes, TeraBytes
  - Time to compress is large (hours)
  - Estimation can run a short time (minutes)
  - Can obtain accuracy guarantees
- Good for:
  - Avoiding compression (and associated overheads) for incompressible volumes.
  - Evaluation and sizing

Micro scale

- Single write
  - KiloBytes
  - Time to compress is small (miliseconds)
  - Estimation has to be ultra quick
  - Heuristic
- Good for:
  - On the fly decisions
  - No prior information about the data
  - Data with varying compression ratio

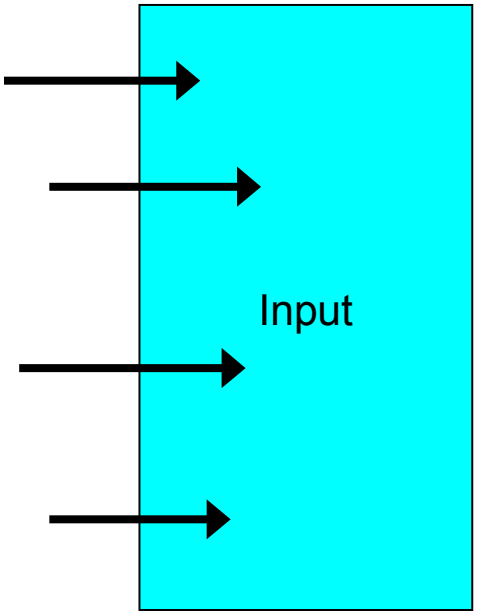
Part I – The Macro

- **Input:** Large volume of data
  - Block volume, file system, etc..

- **Goal:** Estimate the overall compression ratio with **accuracy guarantee**.

The framework:

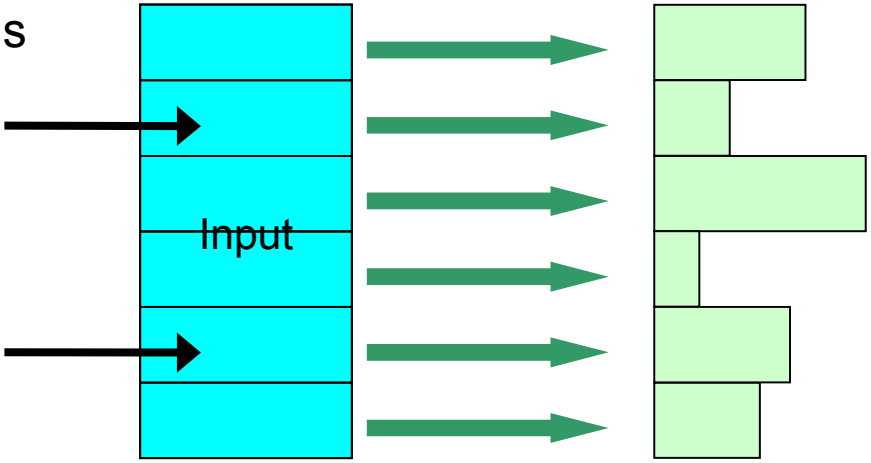
- Choose m random locations
- Compute an average of the compression ratio of these locations



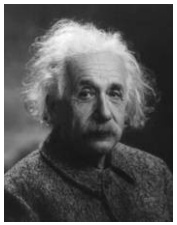
- What is “location”?
- What is “compression ratio of a location”?
- How do we get a guarantee?

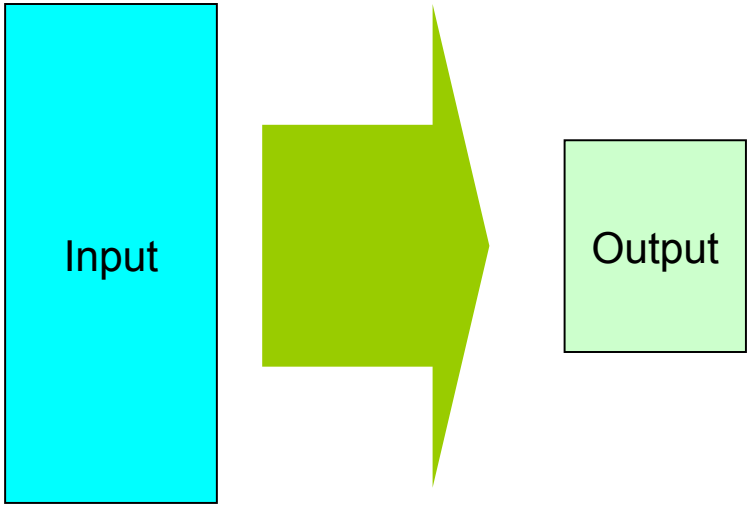
The Macro-scale – what to sample?

- Solution is straightforward if compression is done independently on fixed size chuncks
  - Uniformly sample input chuncks
- But in general this is not the case
  - For example, Zlib can output one zip file for very large files/volumes.



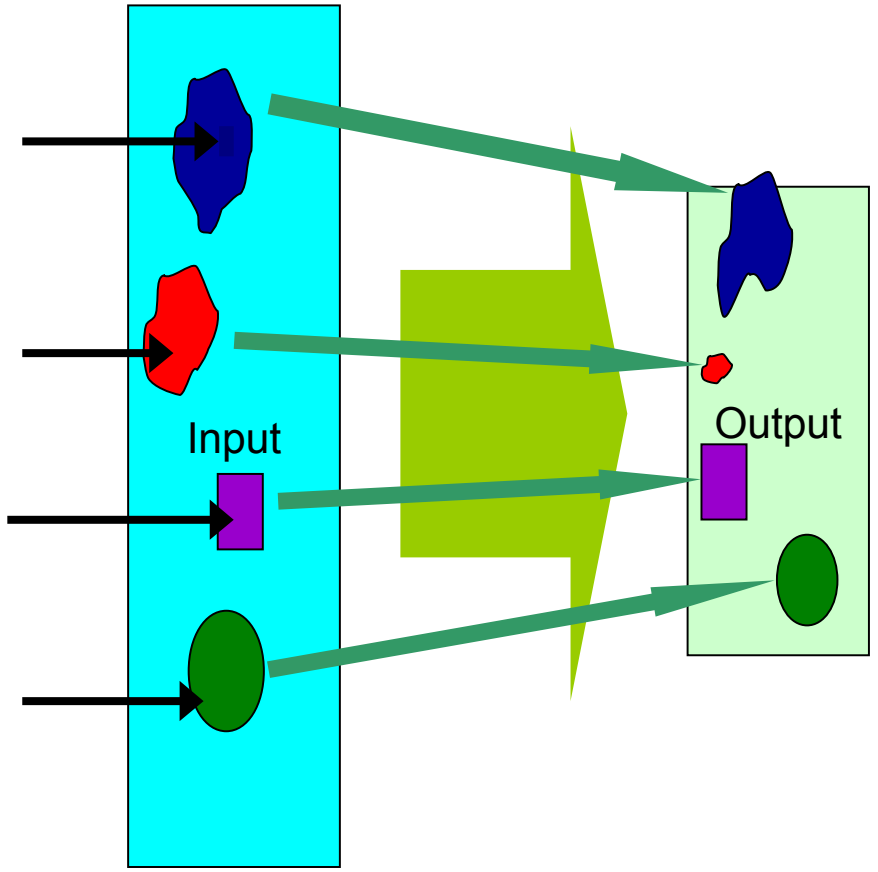
- What about the impossibility result for LZ77?
  - [RRRS07]…
- "In theory, theory and practice are the same. In practice, they are not…"
  - Albert Einstein





The Macro-scale – what to sample?

- Real life implementations of compression algorithms are subject to “locality limits”
  - Don’t want to hold long back pointers
  - Memory management, need to flush their buffers
- We sample single bytes
- Define the **contribution** of a byte as the compression ratio of its **locality**
- Analysis sketch:
  - Prove that the overall ratio is an average of the contributions of all bytes in the input
  - Use known statistical analysis for estimating averages via sampling.

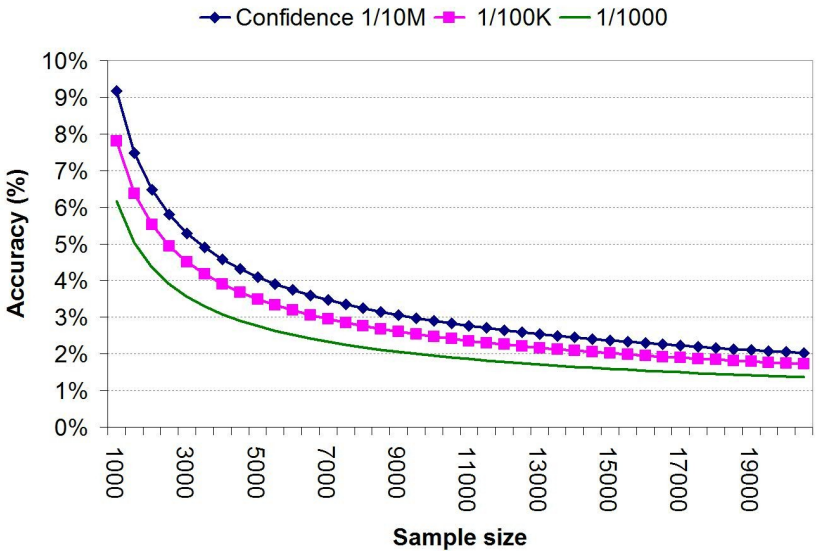


The Macro-scale – Sample size and accuracy

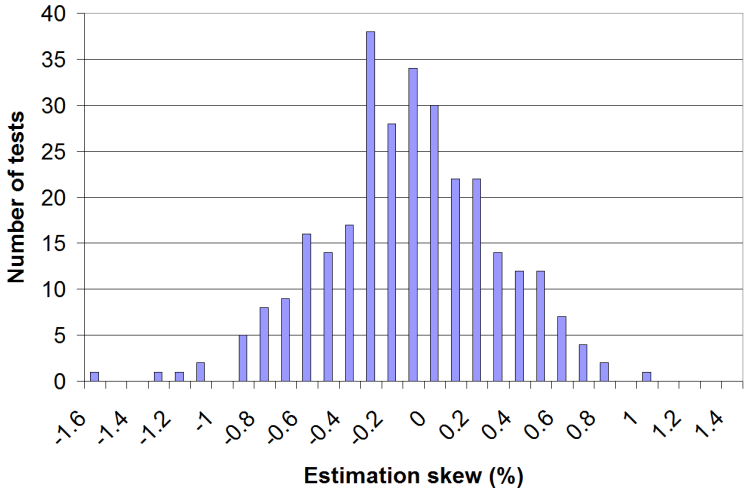
Analysis yields:

$$\text{Confidence} \leq 2e^{-2m \cdot \text{Accuracy}}$$

- Accuracy is a bound on the additive error
- Plug desired confidence and accuracy into equation to get the required sample size
- Sample size independent of Volume size!



- Results of an estimator run are normally distributed around the actual compression ratio
- Width of Gaussian dictated by sample size.



The Macro-scale – the Actual Tool

- Written in C
- Multi-threaded

- Two implementations:

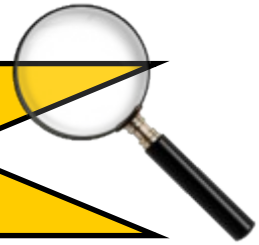
1. IBM Real-Time compression
2. Zlib compression on full objects

- Tested on real life data
- Example of a run: 73 seconds on a 3.2 TB volume – Error ~0.5%
  - Exhaustive run took almost 4 hours

- IBM Comprestimator – the macro-scale for IBM Real-time compression on Storewize V7000 and SAN Volume controller: Downloadable at:

http://www-01.ibm.com/support/docview.wss?uid=ssg1S4001012

Part II – The Micro



- **Input:** A single write
  - For example: 8KB, 16KB, 32KB, 128KB
- **Goal:** Quickly recommend to zip or not to zip.
  - Has to be much faster than actual compression!
    - Don't want to read the entire chunk
    - Impossible to get guarantees – locality is the entire chunk

Option 1: Prefix estimation

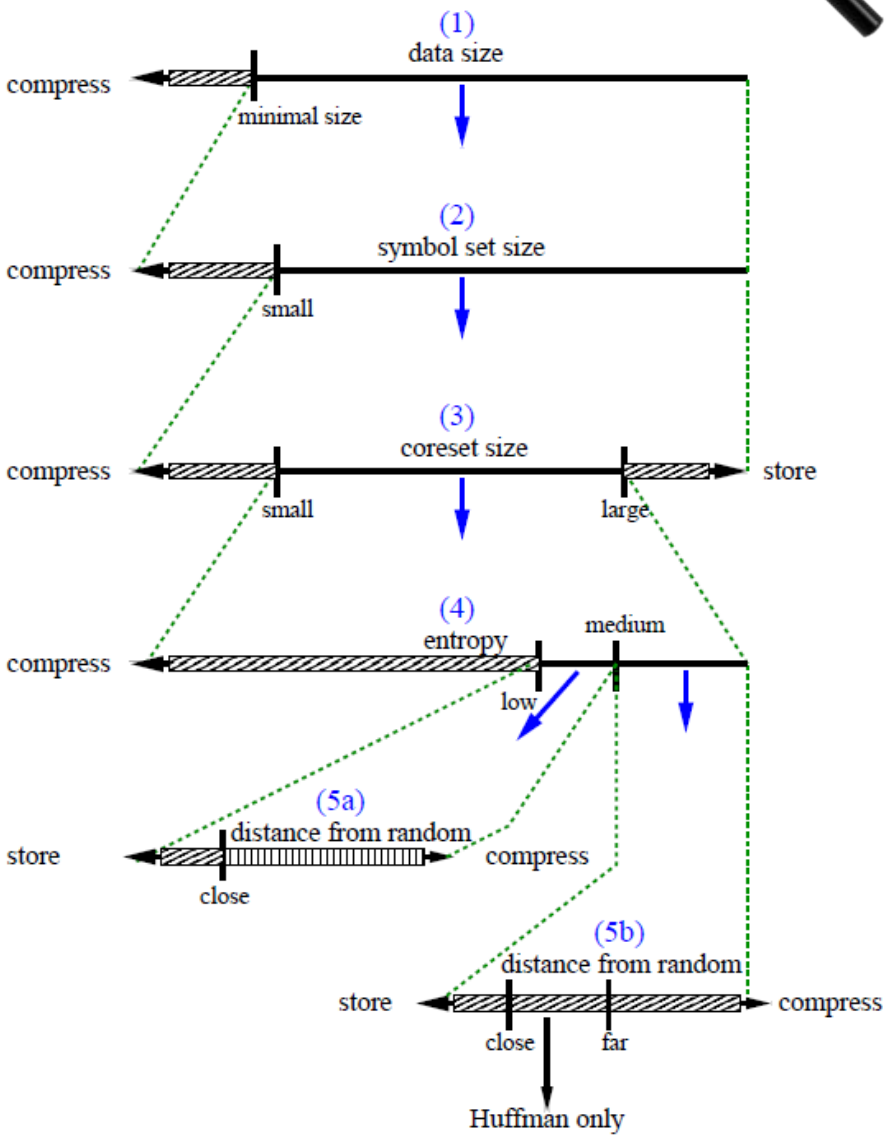
- Start compressing the input chunk and stop early
  - E.g., in a 8KB chunk stop after 1KB
- Evaluate compression ratio thus far:
  - Compressed well → Continue
  - Incompressible → abort and copy the uncompressed chunk
- Good for compressible data – zero overhead ☺
  - Not so much for incompressible data… ☹
- Problematic for data that changes in the middle ☹
  - E.g. A text document with an embedded photo

Option 2: The Heuristics Method

- Collect a set of basic indicators about the chunk

- Based on the indicators make one of 3 recommendations on the chunk:
  1. **Don't compress**
  2. **Compress**
  3. **Huffman** - Skip the LZ compression and use only Huffman encoding

- **Indicators**
  - Core-set size – The character set that makes up most of the data
  - Byte-Entropy
  - Symbol-pairs distribution indicator



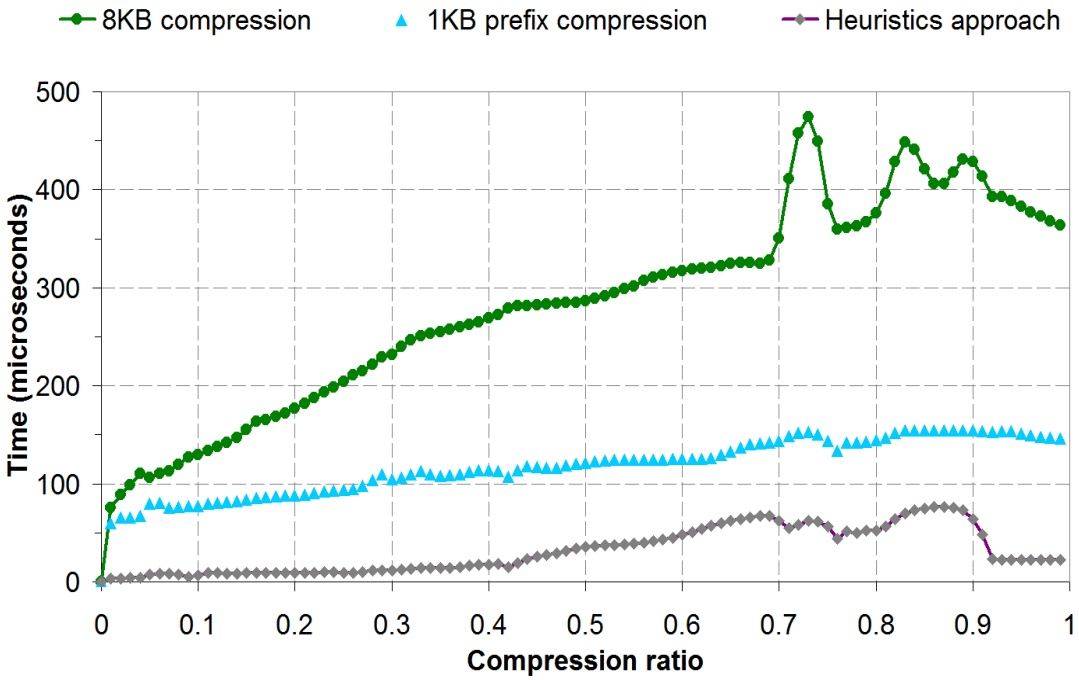
Heuristics Method: Speeding it up

Employed a number of techniques to improve performance:

- **Sampling**
  - Indicators are computed on a **sample** of the data
- **Adaptive sampling**
  - Add more samples until the desired statistical significance is reached.
- **Lazy evaluation**
  - Calculate the heuristics sequentially, in progressing order of difficulty (light to heavy)
    - Attempt to reach a decision as early as possible

Evaluation

- Running time: of heuristics, prefix and full compression
- Measured on a benchmark collection of real life data
  - Tested on mixed data types
  - Over 300 GB
  - 17790 files



- Time vs. compression trade-off:
  - For example, on real life data with 32KB writes, the methods exhibit:
    - Prefix: 74% CPU utilization at the price 2.2% capacity increase
    - Heuristics: 65% CPU utilization at the price 2.3% capacity increase

Putting it all together

Use an a combination of the different methods:

- Whenever applicable – use the **macro-scale**
- After this, the operation mode depends on the observed data:

compressible:  incompressible: 

- When most (or all) is compressible
  → use **prefix** estimation

- When significant percent is incompressible
  → use **heuristics** method

- When most is incompressible
  → **Turn compression off**, run **macro-scale** off-line to detect if a change in tendencies occured

Thank You !