Compression

Making Flash (Even) Cheaper

Thomas Parnell, IBM Research - Zurich
1. Why do we want compression?
2. What algorithms exist?
3. How should we choose an algorithm?
4. How does compression affect controller design?
Why Do We Want Compression?

- NAND Flash technology has seen unprecedented technological innovation over the last few years
  - 1y/z nm, TLC, 3D, QLC
- Despite this, Flash remains a relatively expensive storage medium (compared to consumer HDD)
- Data reduction techniques such as compression and deduplication offer a way to “close the gap”
- Compression can bring additional performance benefits
Compression Algorithms

- Lempel-Ziv
- Huffman Coding
- Arithmetic Coding
- Adaptive Statistical Compression
- Context Mixing
There is a lot of talk about Flash Memory at the Flash Memory Summit

Replace long matches of characters with (Length, Distance) back-pointers

There is a lot of talk about Flash Memory at the (12,20) Summit

Length=12 chars

Distance=20 chars

[Ziv & Lempel, 1977]

Lempel-Ziv coding underpins many compression algorithms widely in use today: LZ4, LZO, GZIP

GZIP uses Lempel-Ziv together with a secondary compression technique.....
Huffman Coding

Data Model

Histogram that can be extracted from input data

Symbol Count

Symbol Index

A  B  C  D
1  1  1  2

Huffman Tree

Input Symbol | Output Code
-------------|-------------
A            | 001         
B            | 1           
C            | 000         
D            | 01          

Variable Length Encoding

Frequent symbols are assigned short codes, infrequent symbols are assigned long codes

[Huffman, 1952]
Example: GZIP

A header is added to the compressed data block containing a description of the Huffman tree that was used to encode the data.

The decompressor must first re-build the Huffman tree from the header and then un-do the two encoding steps.

Uncompressed Input Data

e.g. 32KB

LZ Encoding

Build Huffman Tree

Huffman Encoding

Compressed Data

Header
Arithmetic Coding

Data Model

Whole sequence of input symbols is encoded as an interval

Histogram that can be extracted from input data

One key advantage of Arithmetic coding over Huffman coding: the data model can change during encoding

[Witten, Neal, Cleary 1987]
Adaptive Statistical Compression

Exact same model is generated in comp and decomp. There is no need to store model in header.

Unlike GZIP, we only need 1 pass through the data.
Context Mixing

Mix together the predictions of an ensemble of context models

We can learn which models are good and which models are bad and weight accordingly

Pr($x_i|x_{i-1}, ..., x_1$)

Examples of CM algorithms:
PPM [Cleary & Witten, 1988]
PAQ [Mahoney, 2005]

In PAQ the mixer is implemented as a neural network. This algorithm won the Hutter Prize.
Which is the best algorithm?

- The best choice of algorithm varies depending on application.
- Let us define compression ratio as follows:
  \[ CR = \frac{\text{Uncompressed size of data}}{\text{Compressed size of data}} \]
- CR=2 would result in a 2x increase in the capacity of the system that can be exposed to the user.
- Key trade-offs involved in selecting an algorithm:

<table>
<thead>
<tr>
<th>Compression Ratio</th>
<th>Vs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression Bandwidth</td>
<td></td>
</tr>
<tr>
<td>Decompression Bandwidth</td>
<td></td>
</tr>
<tr>
<td>Memory Usage</td>
<td></td>
</tr>
</tbody>
</table>

Most critical issues for Flash-based storage??
Exploring the Algorithm Space

Squash compression benchmark
IBM data corpus

PAQ algorithm (Context Mixing)
Huge CR but very slow!

Brotli (LZ+Huffman+2\textsuperscript{nd} order context)
[Alakuijala, Szabadka, 2016]

GZIP (LZ+Huffman)
Good CR, decent speed

LZ4 (LZ-only)
Low CR, incredible speed

Pareto Frontier
Flash-specific Considerations

- Integration of compression can have a profound impact on the design of a Flash controller
- Compression can also change performance
- We will consider two interesting examples:
  1. Write amplification
  2. Data placement
Write Amplification (CR=1)

\[
WA = \frac{\text{Total Physical Data Written}}{\text{Total Logical Data Written}}
\]

<table>
<thead>
<tr>
<th>Logical</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB</td>
<td>16KB</td>
</tr>
<tr>
<td>10 pages/block</td>
<td>10 pages/block</td>
</tr>
</tbody>
</table>

- # of logical pages in block: 40
- # of valid logical pages: 30
- # of invalid logical pages: 10 (1/4 of block)
- Logical host writes: 40 * 4KB = 160KB
- Physical host writes: 40 * 4KB = 160KB
- Physical relocation writes: 30 * 4KB = 120KB

\[
WA = \frac{160KB + 120KB}{160KB} = 1.75
\]
Write Amplification (CR=2)

$$WA = \frac{\text{Total Physical Data Written}}{\text{Total Logical Data Written}}$$

# of logical pages in block: 80
# of valid logical pages: 60
# of invalid logical pages: 20 (1/4 of block)
Logical host writes: $80 \times 4\text{KB} = 320\text{KB}$
Physical host writes: $80 \times 2\text{KB} = 160\text{KB}$
Physical relocation writes: $60 \times 2\text{KB} = 120\text{KB}$

$$WA = \frac{160\text{KB} + 120\text{KB}}{320\text{KB}} = 0.875$$

Compression leads to a reduction in WA
Logical pages can straddle physical pages!
Conclusions

• Compression has significant benefits for Flash-based systems:
  - Increased logical capacity
  - Decreased write amplification

• However, it makes controller design more challenging:
  - Increased meta-data requirements
  - Increased complexity of Logical-to-Physical mapping

• Choose the right compression algorithm for your system:
  - What compression/decompression bandwidth is required?
  - How much memory is available?