wflz

Jiri Hamberg

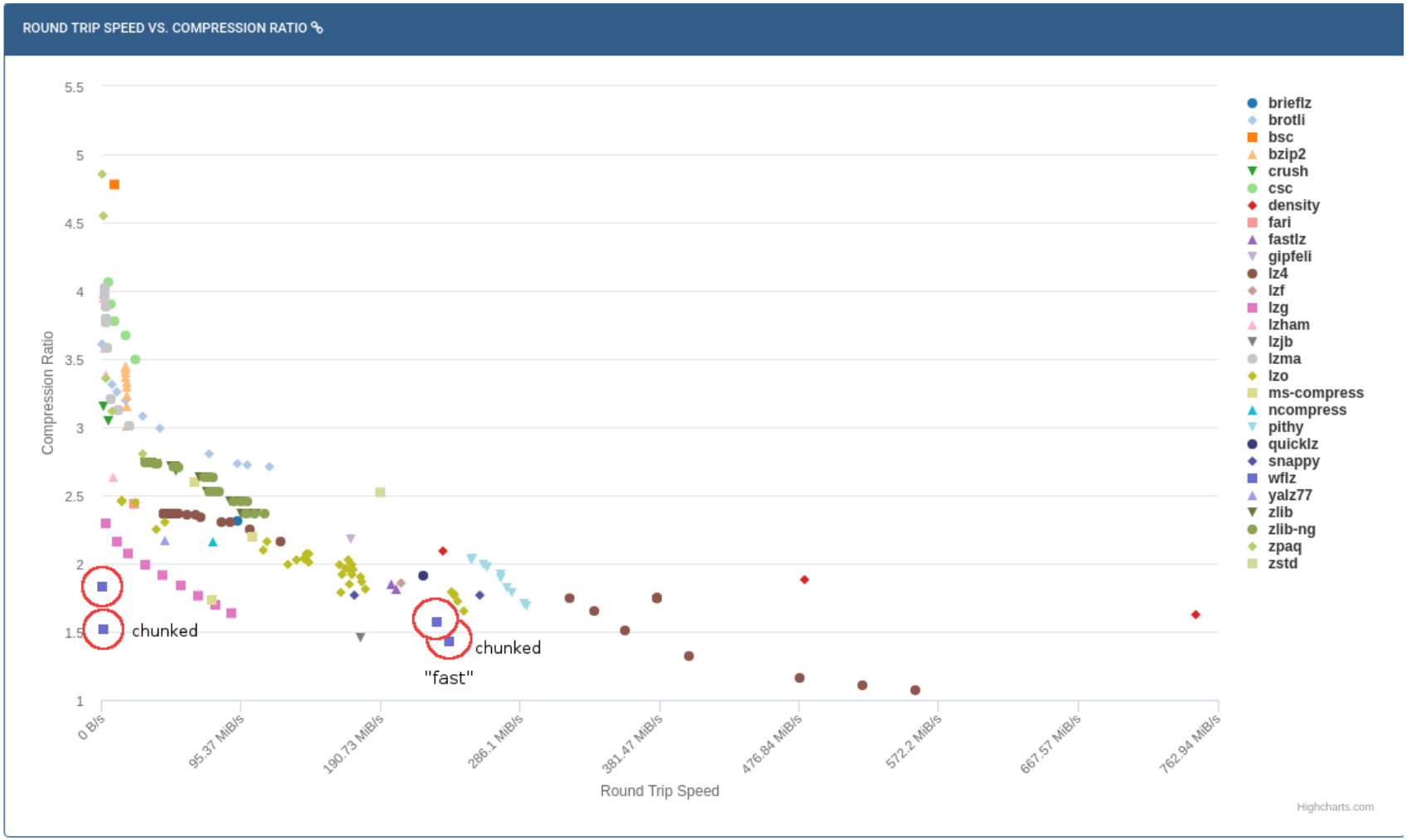
Data Compression Seminar 2017

20.3.2017

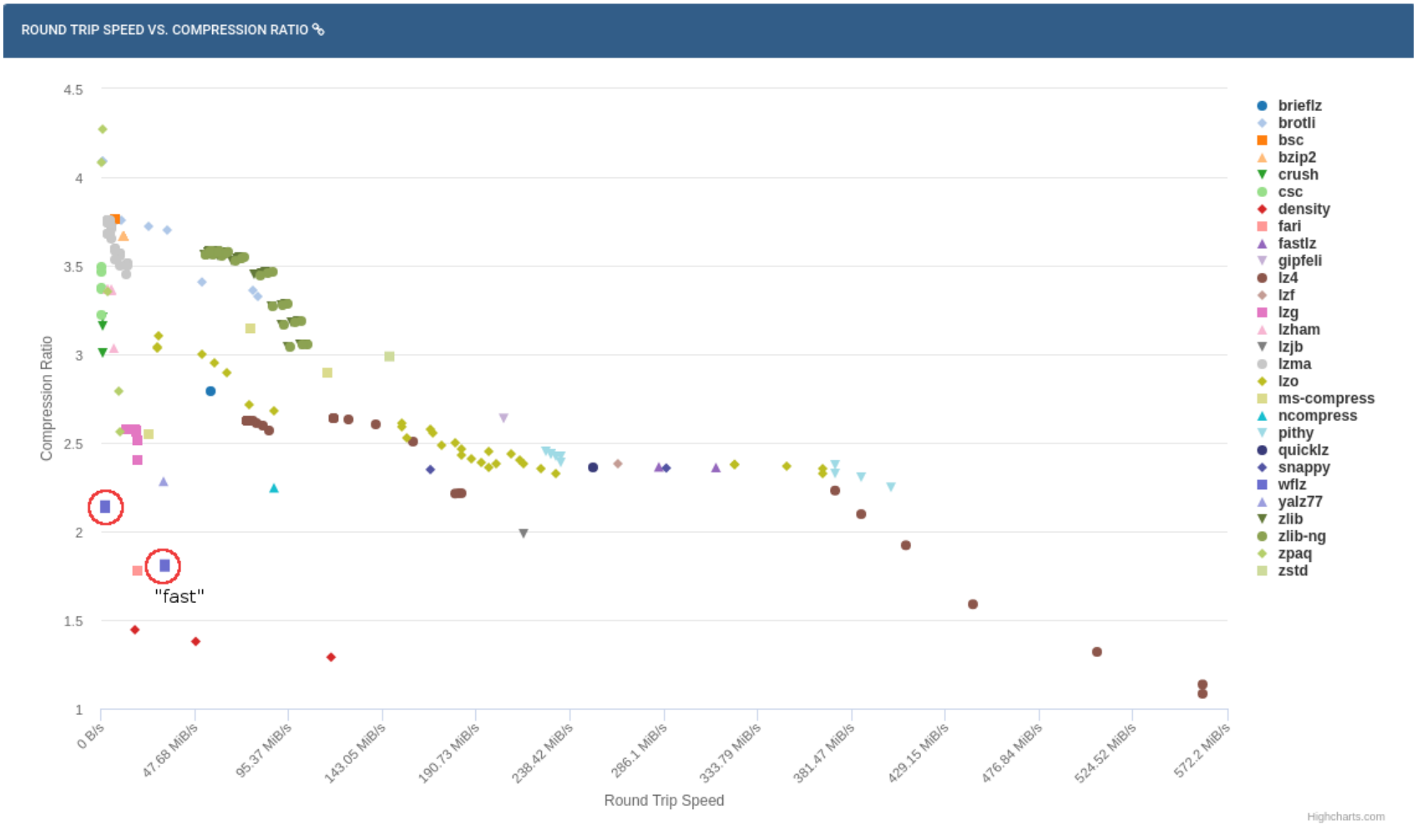
Overview

- Simple open source C library - 837 lines of code
- 2 compression levels - "fast" and normal
- Operates in memory
- Pretty bad performance overall

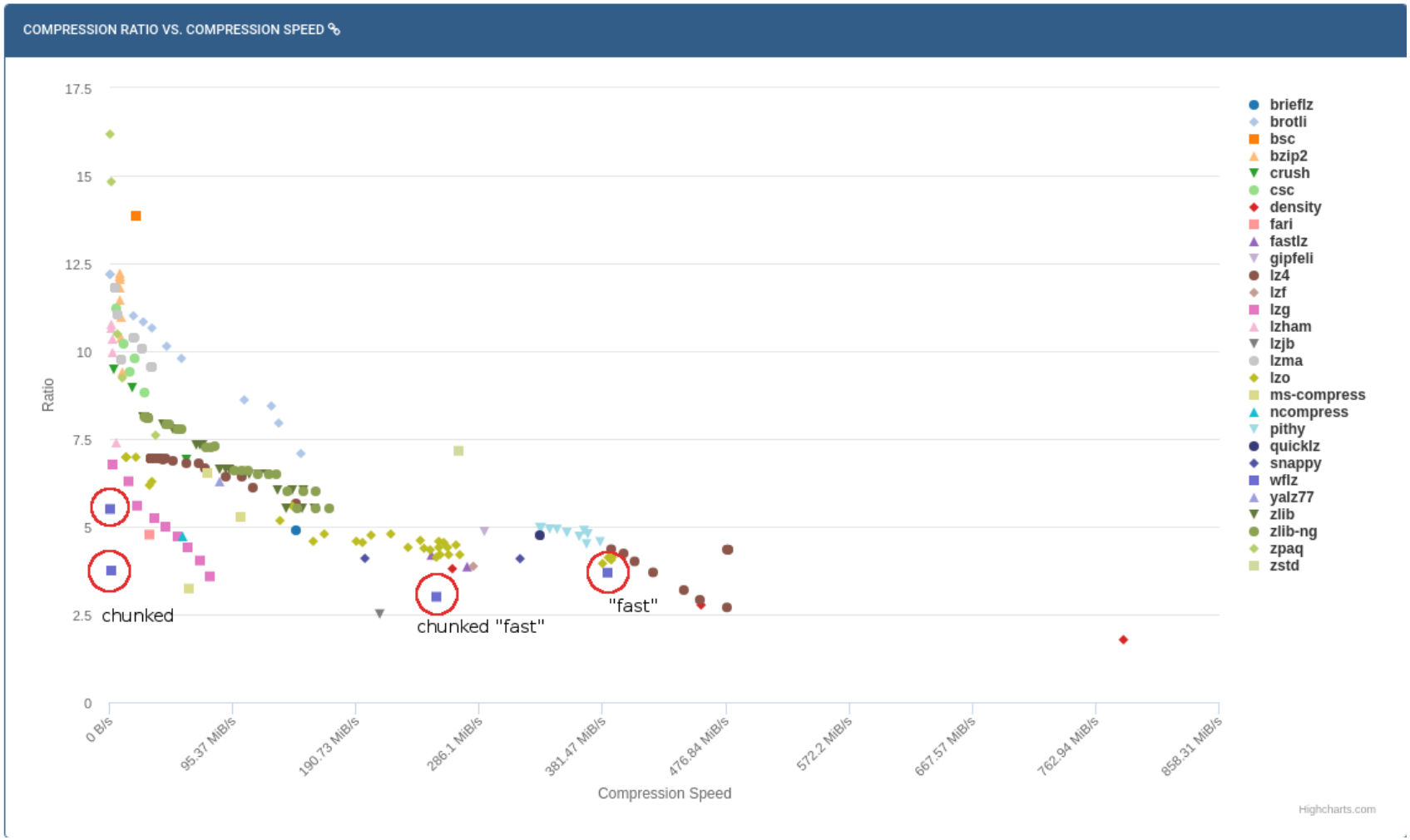
Performance - Wikipedia dump (95.37 MiB)



Performance - C source code (10.89 KiB)



Performance - xml files (5.1 MiB)



Compression Format

- Blocks and literals
- Block consists of **distance** to reference, **length** of the reference and **number of literals** before next block

```
typedef struct _wfLZ_Block
{
        uint16_t dist;
        uint8_t  length;
        uint8_t  numLiterals;  // how many literals are th
} wfLZ_Block;
```

Compression ("fast")

1. Read input one byte at a time
2. Maintain hash table (of next four bytes at every position)
3. Check longest match between *current position* and *hashtable[hash(currentPos)]*
   - 3.1 If match is longer than than block size (4 bytes), and not too far away (distance stored in 2 bytes), write new block
   - 3.2 Otherwise write new literal

Compression (normal)

- Like "fast" compression but instead of only checking longest match at *hashtable[hash(currentPos)]*
  - Find the best match in range

$$[currentPos - maxMatchDist, hashtable[\ hash(currentPos)\ ]]$$

- Guarantees that best match is found, but is very slow
- For example, with dataset "dickens", wflz achieved compression speed 19.5Kib/s and ratio 1.71 whereas the fastest compressor, density achieved compression speed 227.75Mib/s (4 orders of magnitude!) and compression ratio 1.75

```
if( hashPos != NULL )
{
  maxMatchLen = WFLZ_MAX_MATCH_LEN > bytesLeft ? bytesLeft : WFLZ_M
  windowStart = src - WFLZ_MAX_MATCH_DIST;
  if( windowStart > hashPos ) window = hashPos;
  if( windowStart < in ) windowStart = in;

  // now that we have a search window established for our current p
  for( ; window >= windowStart; --window )
  {
    ureg_t matchLen = wfLZ_MemCmp( window, src, maxMatchLen );
    if( matchLen > bestMatchLen )
    {
      bestMatchLen = matchLen;
      bestMatchDist = src - window;
      if( matchLen == maxMatchLen ) { break; }
    }
  }
}
```

Decompression

- Basic LZ-decompression
- Uses some minor tricks to improve copy speed
  - Duff's Device (loop unrolling) (I wonder if this is actually any faster than *memcpy* on most platforms though)

Duff's Device (Yes, this is valid C)

```
ireg_t n = (len+7) / 8;
switch( len % 8 )
{
  case 0: do { *dst++ = *cpySrc++;
  case 7:      *dst++ = *cpySrc++;
  case 6:      *dst++ = *cpySrc++;
  case 5:      *dst++ = *cpySrc++;
  case 4:      *dst++ = *cpySrc++;
  case 3:      *dst++ = *cpySrc++;
  case 2:      *dst++ = *cpySrc++;
  case 1:      *dst++ = *cpySrc++;
  } while( --n > 0 );
}
```

Other tricks

- Use of the *restrict* C99-keyword for compiler optimizations