// Heuristic match skipping: If 32 bytes are scanned with no matches

// found, start looking only at every other byte. If 32 more bytes are

// scanned, look at every third byte, etc.. When a match is found,

// immediately go back to looking at every byte. This is a small loss

// (~5% performance, ~0.1% density) for compressible data due to more

// bookkeeping, but for non-compressible data (such as JPEG) it's a huge

// win since the compressor quickly "realizes" the data is incompressible

// and doesn't bother looking for matches everywhere.

* [Snappy](https://google.github.io/snappy/) is a compression algorithm designed by Google with the goal of providing reasonably good compression in minimal amounts of time. If you look at the usecases on the linked page you will notice that distributed computing (Hadoop, Cassandra, etc) coming up frequently. This makes a lot of sense. These systems often contain clusters of hundreds or even thousands of nodes and data needs to be moved between those nodes as quickly as possible. The speed will generally be a function of the size of the data. So it makes sense that we might want to compress data when sending it between nodes. But if the cost of compression is too high than the 25 milliseconds you saved with 50% compression might have been offset by the 35 milliseconds it took to perform the compression. There is a sweet spot between speed and compression ratios that Snappy is shooting for.
* As the names imply, the library's primary aim is speed. "It does not aim for maximum compression, or compatibility with any other compression library," Google says. "Instead, it aims for very high speeds and reasonable compression."
* Compared to the fastest mode of the popular [zlib compression library](http://zlib.net/" \t "_blank), Google says, the C++-based Snappy is an order of magnitude faster in most cases (roughly ten times faster), but the compressed files are between 20 and 100 per cent larger. Running in 64-bit mode on a single core of a 2.26Ghz "Westmere" Intel Core i7 processor, according to the company, Snappy compresses at roughly 250MB/sec and decompresses at 500MB/sec.
* Google says that the typical compression ratios are about 1.5x to 1.7x for plain text and about 2x to 4x for HTML. zlib in its fastest mode gives you 2.6x to 2.8x for plain text and 3x to 7x for HTML. " So if you want to save space, or want to compress once and decompress lots of times, use zlib (or bzip2, or…). But if you just want to cut down on your I/O, be it network or disk I/O, Snappy might be for you," says Google engineer Steinar Gunderson.
* Virtually all of Google's online service run atop a uniform distributed infrastructure based on the proprietary Google File System (GFS), MapReduce, BigTable, and other platforms.
* While Snappy compression is faster, you might need to factor slightly higher storage costs. Your files at rest will be bigger. If you are charged, as most cloud storage systems like Amazon S3 do, based on the amount of data stored, the costs will be higher. However, the flip side is compute costs are reduced. Why? You need fewer resources for processing the data into a compressed format with Snappy. For example, if you see a 20% to 50% improvement in run time using Snappy vs gzip, then the tradeoff can be worth it. This amounts to trading IO load for CPU load. As a general rule, compute resources are more expensive than storage.
* If the infrastructure starves on disk capacity but has no performance problems it may be logical to use an algorithm that give huge compression ratios, losing some time on CPU but that’s usually not the case. Large capacity disks are far cheaper than fast storage solutions (think SSDs) so it is better for a compression algorithm being faster than being able to give higher compression ratios.

Varint

Format