It’s “Analysis of Algorithms” time, time to sell any stock you are holding a Bubble Sort business.

MurmurHash was the latest big advancement in Hashing. It was published in Austin Appleby’s personal blog. Google hired him and has taken over Murmurhash, and published a variant of it called CityHash. Strangely, many Google publications failed to recognize SpookyHash - published October 31, Ground Hog Day, and other celebrated days, citing only MurmurHash. It is a tragedy that Google shut down what was known as “Google Code”, because it was by far the most comprehensive collection of indexed source code in the world that contained many ancient software projects and inner workings of pretty much everything built with software. [http://stackoverflow.com/questions/**777**8034/replacement-for-google-code-search](http://stackoverflow.com/questions/7778034/replacement-for-google-code-search)

Here are two links to help you quickly catch up on hashing algorithms to get your prerequisites up to date. To cut through all the trending Hash Hype, I recommend this brief overview of hashing.:

<http://www.homolog.us/blogs/blog/2013/05/06/why-computer-science-professors-dislike-hash-functions/>

written from an Oracle DBMS engineer’s perspective. <http://www.burtleburtle.net/bob/hash/doobs.html>

I wanted to see if CityHash could help me speed up my indexing scheme so I decided to test SpookyHash/CityHash within GHash which is part of XMLFoundation. The test counts CPU cycles and/or microseconds on both Windows and Linux while indexing large datasets for both 32 and 64 bit applications. The source code for this test is included in the example program ExIndexObjects.

The results are very interesting. GHash is so fast that CityHash slows it down. The GHash is a unique algorithm designed to index XMLObjects – but it can index anything. In summary, GHash is an Array of B-Trees. It handles hash collisions so efficiently that it eliminates need for "low collision" hashing algorithms such as CityHash which use far more CPU than a simple Rotating Hash, and further research will determine just how low it can go in the simplification of that step.

CRC-n. is faster than CityHash. For Checksumming, or creating a hash function with a perfect distribution (aka avalanche effect) CRC is a better choice. CityHash and SpookyHash are curious works in math that have only 1 possible application – they can destroy data quickly(like the Google Code Search). If you have a HUGE amount of sensitive data and want to delete it (which will mark the disk sectors ‘free’ for the OS to use) and DESTROY it so that it could never be recovered by any disk utility tools. There are many free utilities that accomplish this already, but with these new algorithms the software can accomplish more block corruption in less time. Why does Google Inc invest in and market algorithms with no purpose in any of their active projects? If I was a share-holder I would vote for new management.

GHash ought not be confused with a block cruncher. The name GHashTableTreeStack was too long so it’s called GHash. The long name would be more proper, like Mr. Hash. GHash is one single "Data Structure Algorithm" that combines multiple algorithms and data structures (see highlighted) as parts of the whole. GHash uses a Rotating Hash to index a Static Array of Binary Trees. A GBTree is a variant of an automatically balancing AVL Tree that contains an alternate index so that beyond traverse Ascending and Descending (like any B-Tree via Left/Right) it also has a secondary index via Next/Previous which allows the GBTree to also be traversed in the order that items were inserted. The secondary index can optimize certain bulk commits (aka disk writes) of individual updates that were applied while it was stored in RAM by GBTree. The GHash cannot do that, but it uses a GBTree that can. The XML data updates and application layer updates primarily use only the primary index. Upon a special kind of commit the fragmented memory structure can be flattened back to it’s initial state into the same order that they came from the disk. Another important algorithmic component of GHash is the unique (aka one of a kind) GHashIterator which allows SIMPLE and non-blocking THREAD-SAFE iteration of this complex data structure via an internal integer index into the Static Array used in combination with a GBTreeIterator which is maintains two GStacks that act as LIFO queues of state information used to quickly iterate the Tree portion of this structure, such a task is typically accomplished by a recursive method of the Tree structure. A mere typical approach will force a multithreaded application to block reads of the structure. GHash is non typical and **very** fast.

Each algorithm has various properties. Building from a Window64 starting point, I re-structured the “Google Sparse Hash” project source code. I made major structure changes to the code. I may have made a fix with a hardcoded 48 or the build simplification. I added struct support for Windows, fully redesigned the build, and provide a 64 bit target for Windows which the “Sparse Hash Project” does not. The work is in the file GSparseHash.h. It will be the file of interest in this release, and admittedly it is still a work in progress. It is a starting point.

Although GHash is not a Distributed Data Structure, it is a far more valuable algorithmic component within a “Distributed Data Structure” than CityHash. Just as GHash is made of many algorithmic components so must a structure like BigTable designed and used by Google, which is likely where CityHash is used. BigTable is best defined as a sparse, distributed multi-dimensional sorted map. It is accurately summarized here: <http://www.cs.rutgers.edu/~pxk/417/notes/content/bigtable.html>

Google has been very secretive about the specifics in their design. Perhaps their interest in CityHash is to make more efficient use of the hardware that runs BigTable. Currently, there is little interest in the specifics of the BigTable design due to the superior open source variant of BigTable called HBase which is transactionally more robust than another variant called Cassandra, which Facebook was forced to abandon for undisclosed reasons (after promoting the technology and publishing a comparison between HBase and Cassandra). Cassandra was designed on “Gossip Protocol” which optimizes writes. HBase optimizes reads and Transactional requirements. FIFO queues are difficult (if not impossible) to implement with Cassandra, and admittedly many implementations do not need a FIFO queue for current needs. Technology investment mistakes like what Facebook recently experienced cost time and money. On the other hand, Yahoo is “experiencing a whole new range of use cases for HBase” which they have now several years invested in. Cassandra works for Twitter and they’re not gonna quit ’er any quicker than Google is going to upgrade to HBase when they can buy more hardware and pay higher electric bills. It would otherwise involve an entire re-write of everything they are deeply invested in.

Heavily invested in, the B2B software sector came to the fate of an entire re-write or keep it going with hacks, retrofits, and new failure points because they heavily invested in technologies developed in the years just prior to the 1999 W3C Global Technology Commitment to XML. Immediately, future technology needs and design paths in the EDI and System Integrations categories were clear to all those already in that area. Future technology decisions in companies all over the world happened overnight like a mushroom pops up in the yard. Like how Netscape popped up immediately because Navigator had market adoption. Everything is subject to change. Google wasn’t even founded until 1998.

Here’s a little history about myself in the algorithms. I’m just a college dropout, however I have passed all the required prerequisites during my time in the university. In 1997 I worked at an odd little company in San Francisco composed of 3 people. It was newly founded by Richard Singer who had written the Approach DMBS that was eventually owned by Lotus, then IBM, and became the DBMS under Lotus notes. Bram Cohen was the other person. Bram had some of the first experience with Java when it was new and he single handedly built a nice Java GUI. Bram later invented the Torrent – but not while we 3 were working together building a distributed datamart partitioning product designed for large data warehouse implementations.

I built the ODBC driver which did some tricked out cache replication within the distributed design and worked with Rich on the random access disk persistence. Rich wrote a tricked out SQL parser that converts and re-writes the SQL for the distributed datamart. Our product benchmarked better than the currently leading product of the closest design and target market which was HP’s product called “Intelligent Warehouse”. Microsoft featured our work in the November 1997 Comdex show. OK – enough history let’s turn our attention toward the present and the future.

The large base of investment in many technologies built on design decisions made prior to the significance of XML cumulated into the dot com bust in the first quarter of 2001, as the situation was recognized in various ways and at various times providing many folks time to exit and spread the losses to other investors with less technology insight. It reshaped Nasdaq while the whole industry reshaped during the following years. The bubble went bust for dot com. Wait till you see the dot gov bust. You’ll get it sorted out eventually. I’ve already got it hashed out.

Boom. Bust. Readjust. Ashes to Ashes. Dust to Dust. In God we Trust.