Hashimoto: I/O bound proof of work

Thaddeus Dryja

Abstract: Using a cryptographic hash function not as a proof of work by itself, but rather as a generator of pointers to a shared data set, allows for an I/O bound proof of work. This method of proof of work is difficult to optimize via ASIC design, and difficult to outsource to nodes without the full data set. The name is based on the three operations which comprise the algorithm: hash, shift, and modulo.

The need for proofs which are difficult to outsource and optimize

A common challenge in cryptocurrency development is maintaining decentralization of the network. The use of proof of work to achieve decentralized consensus has been most notably demonstrated by Bitcoin, which uses partial collisions with zero of sha256, similar to hashcash. As Bitcoin's popularity has grown, dedicated hardware (currently application specific integrated circuits, or ASICs) has been produced to rapidly iterate the hash-based proof of work function. Newer projects similar to Bitcoin often use different algorithms for proof of work, and often with the goal of ASIC resistance. For algorithms such as Bitcoin's, the improvement factor of ASICs means that commodity computer hardware can no longer be effectively used, potentially limiting adoption.

Proof of work can also be "outsourced", or performed by a dedicated machine (a "miner") without knowledge of what is being verified. This is often the case in Bitcoin's "mining pools". It is also beneficial for a proof of work algorithm to be difficult to outsource, in order to promote decentralization and encourage all nodes participating in the proof of work process to also verify transactions. With these goals in mind, we present Hashimoto, an I/O bound proof of work algorithm we believe to be resistant to both ASIC design and outsourcing.

Initial attempts at "ASIC resistance" involved changing Bitcoin's sha256 algorithm for a different, more memory intensive algorithm, Percival's "scrypt" password based key derivation function[1]. Many implementations set the scrypt arguments to low memory requirements, defeating much of the purpose of the key derivation algorithm. While changing to a new algorithm, coupled with the relative obscurity of the various scrypt-based cryptocurrencies allowed for a delay, scrypt optimized ASICs are now available. Similar attempts at variations or multiple heterogeneous hash functions can at best only delay ASIC implementations.

[1] Percival, Colin. "Stronger key derivation via sequential memory-hard functions."*Self-published* (2009) https://www.tarsnap.com/scrypt/scrypt.pdf

Leveraging shared data sets to create I/O bound proofs

"A supercomputer is a device for turning compute-bound problems into I/O-bound problems."

-Ken Batcher

Instead, an algorithm will have little room to be sped up by new hardware if it acts in a way that commodity computer systems are already optimized for.

Since I/O bounds are what decades of computing research has gone towards solving, it's unlikely that the relatively small motivation of mining a few coins would be able to advance the state of the art in cache hierarchies. In the case that advances are made, they will be likely to impact the entire industry of computer hardware.

Fortuitously, all nodes participating in current implementations of cryptocurrency have a large set of mutually agreed upon data; indeed this "blockchain" is the foundation of the currency. Using this large data set can both limit the advantage of specialized hardware, and require working nodes to have the entire data set.

Hashimoto is based off Bitcoin's proof of work[2]. In Bitcoin's case, as in Hashimoto, a successful proof satisfies the following inequality:

$$hash\_output < target$$

For bitcoin, the hash_output is determined by

$$hash\_output = sha256(prev\_hash, merkle\_root, nonce)$$

where prev_hash is the previous block's hash and cannot be changed. The merkle_root is based on the transactions included in the block, and will be different for each individual node. The nonce is rapidly incremented as hash_outputs are calculated and do not satisfy the inequality. Thus the bottleneck of the proof is the sha256 function, and increasing the speed of sha256 or parallelizing it is something ASICs can do very effectively.

Hashimoto uses this hash output as a starting point, which is used to generated inputs for a second hash function. We call the original hash hash_output_A, and the final result of the proof final_output.

[2] For Bitcoin's proof of work system, see https://en.bitcoin.it/wiki/Block_hashing_algorithm

Hash_output_A can be used to select many transactions from the shared blockchain, which are then used as inputs to the second hash. Instead of organizing transactions into blocks, for this purpose it is simpler to organize all transactions sequentially. For example, the 47th transaction of the 815th block might be termed transaction 141,918. We will use 64 transactions, though higher and lower numbers could work, with different access properties. We define the following functions:

nonce	64-bits. A new nonce is created for each attempt.
get_txid(T)	return the txid (a hash of a transaction) of transaction number T from block B.
block_height	the current height of the block chain, which increases at each new block

Hashimoto chooses transactions by doing the following:

hash_output_A = sha256(prev_hash, merkle_root, nonce)
for *i = 0 to 63* **do**
    shifted_A = hash_output_A >> i
    transaction = shifted_A mod total_transactions
    txid[i] = get_txid(transaction) << i
end **for**
txid_mix = txid[0] $\oplus$ *txid[1] ...* $\oplus$ *txid[63]*
final_output = txid_mix $\oplus$ (nonce << 192)

The target is then compared with final_output, and smaller values are accepted as proofs.

The initial hash output is used to independently and uniformly select 64 transactions from the blockchain. At each of the 64 steps, the hash_output-A is shifted right by one bit, to obtain a new number, shifted_A. A block is chosen by computing shifted_A modulo the total number of blocks, and a transaction chosen by computing shifted_A modulo the number of transactions within that block. These txids are also shifted by the same amount as the shifted_A which selected them. Once the 64 txids have been retrieved, they all XORed together and used as the input for the final hash function, along with the original nonce. The original nonce, shifted up into the most significant bits, is needed in the final XOR function because very small sets of transactions may not contain enough permutations of txids to satisfy the proof of work inequality. In fact, this algorithm only becomes I/O bound as the blockchain expands in size. In the extreme case of a blockchain with only 1 block and 1 transaction, the entire 64 iteration process can be omitted, and the nonce for final_output can be rapidly iterated as the txids will always be the same. With larger blockchains, inclusion of the nonce in the final hash may no longer be necessary but neither is it detrimental.

Analysis of bottlenecks

This method has an exponential tradeoff between hash operations and memory access. Given a blockchain of 100 blocks, if a mining node were to lack even one block, each initial hash would have a probability of $0.99^{64} \approx 0.5$ of being able to retrieve the inputs for the second hash function. Thus a 1% reduction in I/O requires a 2X increase in hash power. A 2% reduction in I/O needs roughly a 4X

increase in hash power, and so on. Holding half the blockchain will allow a miner to get to the second hash operation once for every 10^{20} first hash operations. Clearly, miners need to have the entire blockchain to mine at all.

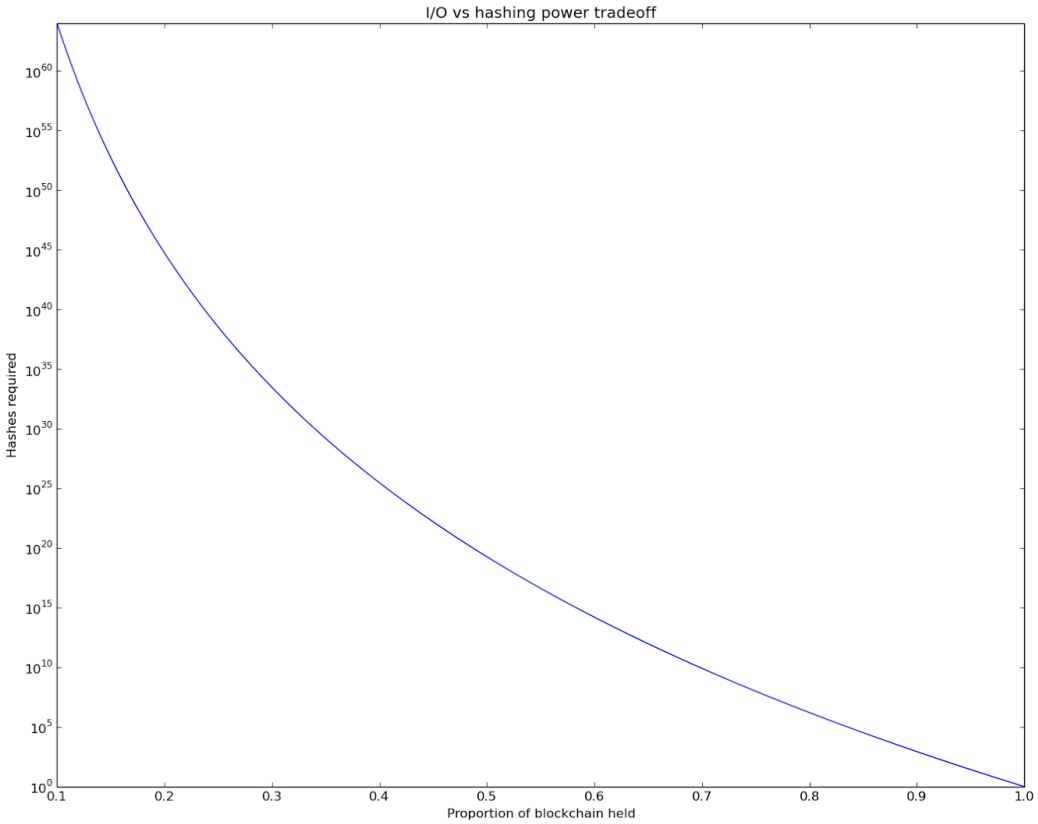

Figure 1. Hashes needed vs proportion of blockchain held. Note the log scale.

This is also easy to verify for all nodes. Receiving nodes need only perform 1 sha256 operations, 128 shift operations, 64 XORs, and several hundred I/O operations. Verifying this proof of work is much less costly than the signature verification that is also needed during block verification.

This method cannot be effectively outsourced. If a server were to host the blockchain for remote mining nodes, the network latency would completely overwhelm the time per hash operation. In this scenario, miners would hold a lookup table of the number of transactions in each block, so that they could compute the hash, shifts, modulo and find each transaction they need the txid of. Sending this to the blockchain host server would be at most a few kilobytes. The blockchain hosting servers response would also be 64 txids * 32 bytes each = 2Kbytes or slightly more. If these requests were serialized, the latency of each transfer is on the order of milliseconds, which would create a limit of under one thousand hashes per second. If instead, miners created large batches of txid requests, each possible hash would require 2Kbyte of data over the network. Running one billion sha256 operations per second, something easily achievable with current hardware, would require network throughput of several terabytes per second. For

a blockchain smaller than a terabyte, the entire blockchain would need to be transferred multiple times per second. Clearly, it would be much faster for each mining node to maintain a local copy of all the data it needs to every possible the proof of work.

Developing an ASIC for such an algorithm would be unlikely to lead to significant improvements over commodity computer hardware. As evidenced by the increase in the Bitcoin network's difficulty, sha256 can be optimized through the use of dedicated hardware. However, Hashimoto uses only 1 sha256 operation per attempt, each of which takes on the order of microseconds for a current CPU. Though in customised hardware a shift can be had for zero time and transistor cost, the shift operation is also very rapid on general purpose CPUs and not a bottleneck. Similarly while modulo operations can also be optimized, the are very rapid on CPUs as well. The bottleneck for Hashimoto is the get_txid(T) function, which requires a read operation to disk or RAM. These I/O operations cannot be effectively cached, as each hash_output_A points to a completely different set of transactions.

The most effective computers for this proof of work will have large amounts of high speed memory. As the hash, shift and modulo operations are fast, many (block, tx) pairs can be computed and queued, and retrieved either sequentially or in parallel. While the blockchain is small, CPUs will be able to cache the entire blockchain on-chip, but as it grows in size moving it DRAM will become necessary. Moving the txids onto a disk will be much slower, and would likely cause a miner to become uncompetitive with miners able to store the entire blockchain in DRAM. If the blockchain grows larger more quickly than DRAM storage, multiple servers using infiniband or another low-latency, high throughput networking system. This type of access pattern is commonly seen in many high performance computing applications, and decades of research has optimized computer hardware for this purpose.

Conclusion

An ASIC resistant, non-outsourceable proof of work can be created by leveraging the large shared data store inherent to cryptocurrencies -- the blockchain. Using a cryptographic hash algorithm to pseudo-randomly select elements from the large shared data set allows the proof of hashing capacity to become a proof of I/O capacity, something current computer hardware is already optimized for. This I/O bound algorithm also ensures that each node contains the entire data set, limiting centralized, outsourcing pools. Hashimoto can a new cryptocurrency to launch in a fair and decentralized way on commodity computer hardware.