Gridcoin Oracles:
The Gridcoin Statistics Scraping and Reporting System ("Scraper")

James C. Owens

Executive Summary

The prior .NET Visual Basic based distributed computing statistics gathering system, referred to as the Gridcoin "Neural Network," was not really a neural network. It was actually a rules based system for gathering 3rd party distributed computing statistics (currently from BOINC projects) off blockchain, summarizing and normalizing them, and then providing a mechanism for the nodes in the network to agree on the statistics and put them on the blockchain in summarized form once a day. (This is referred to as a superblock.) The research rewards are then calculated and generated for/by staking wallets that perform research via the distributed computing platform BOINC, confirmed by other nodes in accordance with blockchain protocols (referred to as Proof-of-Research).

This previous system had a number of serious defects and had been in need of replacement for some time. In October 2018 the author began a project which originally had the goal of fully implementing Paul Jensen's prototype statistics proxy program ("scraper"). (See https://github.com/gridcoin-community/ScraperProxy). As this project progressed, it became apparent that this was more properly scoped as a complete rewrite of the old "Neural Network" subsystem, and should be written entirely in C++ as part of the core wallet. The scraper has been developed on the integrated_scraper branch in the author's Github repository fork of Gridcoin and was merged into the development branch of the official Gridcoin repository on February 25, 2019. (See https://github.com/jamescowens/Gridcoin-Research and https://github.com/gridcoin-community/Gridcoin-Research/commit/989665d699fb9753cd2d519c39ed347d4298652f). It was put into production (mainnet) in the Denise milestone leisure release on May 10, 2019. (See https://github.com/gridcoin-community/Gridcoin-Research/commit/01c88b04dbd44611efc241a451a04206448267f4.)

The new Scraper consists of three major parts:

1. The actual scraper handles the downloading of stats files from the BOINC projects, and the filtering, compression, and publishing (with hashes and signatures) of the stats files to the network. This was designed and written by Jim Owens (“Scraper”)
2. The scraper networking code uses the wallet messaging system in an elegant fashion to automatically distribute the compressed, hashed, and signed stats files to all of the nodes. The author is grateful to Tomas Brada (tomasbrod) for writing a very elegant approach for this part. (“Scraper Net”)
3. The interface to the "neural network", which interfaces the core wallet to the scraper and together with the existing functions in the core wallet, provides the core “neural network” functionality. The author is grateful to Marco Nilsson (ravon) for this contribution. (“NN”). **[Note: In the Fern release, this “shim” was replaced with native interfacing to the beacon registry, researcher class, superblock, and accrual system.]**

Old vs. New Scraper Comparison

Category	Old VB .NET	New Native C++
Scalability	Severely limited. No support for removal of team requirement.	High - At least 20x current capacity - while maintaining constant low load on BOINC statistics sites. Fully supports the removal of the team requirement, which is scheduled for the Elizabeth Milestone (4.1.0.0).
Cross Platform Compatibility	Windows only - Requires GUI.	Completely cross platform - supports all platforms the wallet supports - currently Win64, Win32, Linux (Intel 64 and 32 bit, ARM 64 and 32 bit), and MacOS (Intel 64 bit) and can be run daemon-only (headless).
Reliability/Availability	Low - due to single point of failure for old scraper	High - Support for multiple scrapers, cross-verified by the nodes, with a configurable (nominally 48 hour) statistics retention period, ensures scraper outages are transparent.
Security	Poor - Single scraper model allowed the possibility of a	Very High - Each scraper must be authorized to publish

	man in the middle attack	statistics to the network. Each scraper hashes and signs all statistics and these hashes and signatures are checked and cross-verified by all nodes. Unauthorized scrapers' statistics are deleted and they are banned from the network.
Network Bandwidth Use	High - the original scraper simply forwarded uncompressed and unfiltered statistics files (>300 MB for a complete set), the same as when the nodes downloaded them directly	Extremely Low - the new scrapers download the stats, filter, and compress them, reducing >300 MB of statistics to 4-5 MB for a 48 hour retention period. Statistics are shared in two stages: the statistics directory is "pushed", and then the actual statistics are "pulled" by the nodes to get the statistics the node does not already have. This minimizes network traffic. Since the messages are signed, they can be forwarded by intermediate nodes, just like other network messages, such as transactions.
Client CPU Use	High - the "Neural Net" on each node could eat up at least 1 CPU for up to 30 minutes for processing the statistics.	Extremely Low - the normal nodes process the scraper statistics in under three seconds for a typical Intel CPU. This ensures the CPU goes towards computing not administration.
Client Disk Use	High - up to 2 GB used on the client drive. Significant disk loading during operation	None - All scraper statistics are compressed and stored in memory
Client Memory Use	Moderate. The .NET runtime adds overhead to the wallet	Low - Very little additional memory required (<50 MB).
BOINC Server Resource Use	High - The old scraper sometimes downloaded	Low - Typically five scrapers in operation - each

	statistics files over and over that were already downloaded. If the single scraper was down, each node would fall back to downloading its own statistics, crushing the BOINC servers (250+ nodes at once).	downloads statistics files for a 4 hour window before the superblock is due, only downloading changed statistics. This results in a constant, low load on the BOINC servers only during the 4 hour window regardless of the size of the Gridcoin network.
Maintainability	Low - Used non-native development and build tool chain (Microsoft Visual Studio .NET) that is not open source and also does not play well with core wallet. This hampered development, testing, and the release process.	High - Written to conform to Gridcoin's coding standards and 100% C++, well commented, with a modular design that is easily extensible, and completely integrated into core wallet.

The scraper code is already designed to dispense with the Gridcoin BOINC team requirement and instead use a "consensus beacon list" derived from the appcache list of active beacons to filter the stats by CPID. The code supports using a whitelist of teams allowed in addition to the filtering by CPID, if required by network protocol. Due to security concerns around the beacon advertisement and renewal, until CustomMiner's *Public-key cryptographic proof of account ownership PR #2965* (now accepted by BOINC) is implemented by the projects, or an equivalently strong method of verifying a BOINC account holder is implemented by the Gridcoin wallet, the team requirement via a single entry to the team whitelist, "Gridcoin", will remain in effect.

Security has been designed in from the outset…

The scrapers have two levels of authorization to operate. The first level, controlled by IsScraperAuthorized(), is whether any node can operate as a "scraper", in other words, download the stats files themselves. That does NOT give them the ability to publish those stats to other nodes on the network. The second level, which is the IsScraperAuthorizedToBroadcastManifests() function, is to authorize a particular node to actually be able to publish manifests (essentially a "directory listing" of statistics objects) to other nodes. The second function is intended to override the first, with the first being a network wide policy. So to be clear, if the network wide policy has IsScraperAuthorized() set to false then ONLY nodes that have IsScraperAuthorizedToBroadcastManifests() can download stats at all. If IsScraperAuthorized() is set to true, then you have two levels of operation allowed. Nodes can run -scraper and download stats for themselves. They will only be able to publish manifests if for

that node IsScraperAuthorizedToBroadcastManifests() evaluates to true. This allows flexibility in network protocol without having to do a mandatory upgrade.

The networking code will not allow the acceptance or retention of manifests from nodes that are not authorized to publish, and will ratchet up banscore for those unauthorized nodes, quickly extinguishing them from the network. This prevents flooding the network with malicious scraper attacks and prevents the gaining of control of the consensus by a bad actor. Each scraper is intended to be authorized with a specific private/public key combination, that will be injected by signed administrative message of the scraper's address (equivalent to the scraper's public key) into the appcache. Only those manifests with a public key/signature that match the approved address will be accepted by any node. This security code operates by necessity on both the send and receive sides of the stats (manifest) messages. Normally a node that is not authorized will not attempt to send manifests, but we have to check on the receive side too (similar to the connectblock/acceptblock for blocks) in case someone maliciously modifies a node to send manifests without authorization. Nodes receiving unauthorized manifests by a malicious scraper will automatically delete those manifests and ban the sending node.

We also have to correctly deal with the deauthorization of a previously authorized scraper by the removal of the authorization key (address). In this case the nodes on the network will be in possession of manifests that were previously authorized and now need to be removed. This removal is accomplished during the housekeeping loop to automatically remove existing manifests from scrapers that have been deauthorized.

The scraper system has been designed to operate in a trustless environment. This means the following: a minimum of 2 independent scrapers must be up and publishing manifests for the stats to be accepted. The scraper implements the idea of a "convergence", which is similar to "quorum" that occurs at a later stage in the SB formation. Convergence on the stats means that a minimum of 2 scrapers or the ceiling of 60% of the scrapers that are actively publishing (whichever is greater), agree on the statistics. Ideally, three or more independent scrapers will be operating and "publishing". (Five is probably the ideal number.) The statistics are downloaded in a loop on the scraper nodes that can be set for a specific start time before the need of a SB (nominally 4 hours before a SB is due), and will run continuously in a loop with a nominal 5 min sleep between runs until a SB is formed. The scrapers check the Etags of both team and user statistics files and do not redownload files already downloaded. Since the team IDs corresponding to the whitelisted teams assigned by each project server do not change once assigned, the team IDs are stored on disk to eliminate checking the team files entirely for projects where the team IDs have already been determined. There is a sophisticated file download and retention mechanism, with a default retention period of 48 hours. The scrapers can use a password file which contains usernames and passwords to access sites needing authentication for stats downloads. (This is the solution to the problem of BOINC project statistics sites that need authentication for access, such as Einstein@home.)

Each scraper forms manifests from the downloaded set of compressed scraper files. This is a compound object consisting of metadata, an inventory map, and pointers to an independent parts collection, where each part is essentially a compressed stats file turned into a binary object (BLOB). Each part, and the manifest as a whole, is hashed using the native (double) SHA256 hashing algorithm, and the manifest is signed by the scraper using a designated key in the wallet (authorized by the network protocol) when published to the network. The networking code AUTOMATICALLY propagates the manifests to all nodes using the normal wallet messaging infrastructure. The receiving nodes then deserialize the received manifest inventory, check the signature and form of the manifest for validity and authorization, and then request the requisite compressed part objects from the sending node. Once the receiving node receives all compressed parts for the manifest, the integrity of the parts is verified by comparing the hash of the part with the part hash in the signed, verified manifest. This manifest and its referred to parts are now useable on the receiving node to form a convergence once the manifests and parts from multiple scrapers have been received sufficient to meet the convergence rules. This ensures absolute integrity in the delivery of the stats between the scrapers and the nodes.

The nodes overcome the need to "trust" the scrapers by the exertion of the "convergence" requirement to be able to construct a converged set of stats using the convergence rules mentioned earlier. This means each node will cross compare, using the native hashes, the imprint of the stats objects from each scraper and make sure they agree before using them. This drastically reduces the probability of there being a man in the middle problem or source corruption problem with the scraper stats, since the intent is for each scraper in production to be hosted by an independent host, and the probability of a simultaneous attack that would result in the identical corruption of 3 or more *authorized, independent* scrapers (if there are 5 running) in such a way to make the hashes match, and pass the signature check, is extremely unlikely.

Because each scraper publishes a series of manifests to the network which consists of snapshots of the statistics files every time a statistics file is changed during the 4 hour window leading up to the SB formation, and these are retained for a nominal 48 hours, there will be a map of manifests available from each scraper with which to try to match between scrapers. Once a node has received the appropriate manifests, and done the cross compare (which is done from manifests across scrapers in reverse time order from latest to earliest) and determined that a "convergence" exists, the nodes will then form a trial SB contract upon demand by the appropriate functions in main.cpp (or the NN loop in the scraper) as appropriate.

This contract then operates essentially as the existing NN does, with the hash going in the quorum popularity, etc. Pretty much from this point on, the operation is identical to the existing NN, without all of the baggage.

Some advantages of the new design:

1. Since a low (single digit) number of scrapers can effectively supply the network with statistics, solves the Gridcoin scalability problem, while drastically reducing the load on

the BOINC project servers from the current levels, with near constant low load thereafter regardless of the scale of the Gridcoin network.
2. Entirely native C++ and cross platform. Allowing core development to use a simplified development platform, easier, more transparent debugging, and a smaller, simpler installation footprint. Both the scraper and normal (non-scraper NN) nodes will operate on all targets for the gridcoin wallet. Linux (Intel, armhf, arm64), Windows, and Mac. This includes headless operation as a daemon without a GUI, since the GUI is not necessary.
3. Once the BOINC stats XML files are downloaded, they are filtered and reduced to csv files on the scrapers, which is far more appropriate and efficient for flat data structures like this. The files are gz compressed and uncompressed using boostio gzip compression filters.
4. Security is designed in from the outset, with multiple scrapers required for cross-verification so that the scrapers do not have to be trusted entities (see the more detailed discussion below).
5. The total data storage required in the data Scraper subdirectory for 48 hours of stats files for mainnet is on the order of 4 to 5 MB. (This is opposed to something like 2 GB on the existing NN VB code.)
6. The total data on-disk storage required for normal (non-scraper) nodes is zero. The scraper code has the ability to decompress and process compressed stats objects in the manifests directly in memory without going to disk first. The in memory requirement for the manifests is on the order of about 1 MB per manifest. (And if the manifests have parts in common they are not repeated, because they are referenced.) Also it is worth noting that normal nodes use a large portion of the scraper code for neural network operation. The scraper code itself was designed to be used in different modes by both the scraper nodes and normal nodes.
7. The total processing time to form the complete statistics map and create a SB contract on a normal node once a convergence is formed takes about 5 seconds for mainnet on an average Intel box. (This reduces the CPU usage of NN enabled nodes drastically from today.) This is for ~1900 active beacons, and ~15000 statistics elements across 23 whitelisted projects as of December 2018. This level of performance should easily support scaling the Gridcoin network by a factor of 10 or more without any trouble, and with NO additional load on the BOINC sites.
8. Gets rid of a lot of the old, bizarre snap to grid stuff and other oddities in the old NN code.
9. Has the core structures in place to support conversion to a TCD based approach, although that will require additional development.
10. Has the ability to support either no team filtering, or filtering by a team whitelist, based on the needs of the network.