Universal Miner System Design

DESIGN DOCUMENT

Dave Brown | 5/16/2018

Table of Contents

Overview

Mining cryptocurrencies is currently a popular way of verifying each transaction within the blockchain used against a double spend. Typically a 'proof-of-work' algorithm is used such as described by [1].

Several miners exist which offer dual mining of different currencies and/or on different types of Graphical Processing Units (GPU) such as those sold by NVIDIA and AMD. For example, the Claymore Miner [2] is an example of this type of dual miner. However, it is unclear how flexible the current mining systems are as they appear to be locked into the current currencies and GPU's that the support at the time their product is installed.

The Universal Mining System offers a much more flexible system that optimizes and tunes the overall mining operations making sure that each GPU use operates at peak performance and does so during the maximum amount of time per time period (e.g. day, week, year) specified by the user. And these optimization features are offered all the while supporting one or more miner(s) for numerous different mining operations. For example the Universal Mining system may mine a single currency during one period, and then mine a different currency during a different period. Or, the Universal Mining system may mine two different currencies at the same time if system resources allow for it. Additionally, the Universal Mining system can perform 'mining like' operations other than just mining a cryptocurrency. For example, the Universal Mining System can solve artificial intelligence problems (AI) much in the same way that it mines a cryptocurrency and do so using the same GPU resources also used for mining – we call this "AI mining". The main goal of the Universal Mining System, is to offer a plug-n-play system that allows for 'mining' a solution for any problem that uses computing processing power or even input from users who use their own intellectual 'processing power' to solve a given problem.

HISTORICAL PERSPECTIVE

In 1999, when the Internet was just getting started, researchers at the University of California started a program called SETI@Home [3] that was geared to take advantage of excess computing power that volunteers would donate. Using a large number of computers interconnected across the internet gave the researchers a much larger computing capacity to perform their search for extraterrestrial life.

Also around this time, component models that offer a highly touted plug-n-play functionality had been around for some time that support the "development of replaceable components." [4]

The introduction of Bitcoin [5] in 2009, opened the eyes of many in that it allowed individuals to make transactions in a manner independent of the large financial institutions that traditionally control, and profit, from each and every transaction. The advancements of the blockchain and smart contract make the Universal Resource System possible for they

offer the security, anonymity and transaction support necessary to facilitate transactions with anyone in the world, and do so at the speed of light.

1999 SETI@Home

According to [3], SETI@Home started in 1999 when researchers at the University of California released a citizen-science program called SETI@Home. The program took advantage of excess computing power that volunteers were willing to contribute to the project designed to search for extraterrestrial life. Today, University of California, Berkeley offers a way to participate in SETI related project via their BOINC [6] software that is available for download at https://boinc.berkeley.edu/download.php.

2002 Replaceable Components

"Several popular component-based standards have emerged recently, including JavaBeans® and Enterprise JavaBeans® from Sun Microsystems and the Component Object Model from Microsoft. These component models are being adopted for use in software development as they eliminate opportunities for architectural mismatch and are supported by standard services." [4]

2009 Bitcoin - Blockchain

According to Wikipedia, "Bitcoin was invited [5] by an unknown person or group of people under the name Satoshi Nakamoto and released as open-source software in 2009." [5] The introduction of Bitcoin unleashed a flurry of other cryptocurrencies. Each of these currencies leveraged the fundamental Blockchain idea introduced by Bitcoin. In addition to the Blockchain, Bitcoin introduced the idea of 'mining' which is a cryptographic algorithm to verify each transaction through a computationally intensive proof-of-work algorithm [1] [7] [8] [9].

2013 Ethereum – Smart Contracts

One of these currencies introduced by Vitalik Buterin in 2013 called Ethereum [10], advanced the ball by adding to the decentralized ledger of the Blockchain the idea of what is called a smart contract. Smart contracts were initially introduced by Nick Szabo in his document "Smart Contracts: Building Blocks for Digital Markets." [11] The basic idea prosed in his document is that "[t]he basic idea of smart contracts is that many kinds of contractual clauses (such as liens, bonding, delineation of property rights, etc.) can be embedded in the hardware and software we deal with, in such a way as to make breach of contract expensive (if desired, sometimes prohibitively so) for the breacher." The idea of smart contracts was expanded further allowing new currencies to offer value through a utility that they offered. Several example utility coins include Sia and Golem [12]. Smart contracts are built using the Solidity [13] language.

2014 Sia Coin – Simple Distributed Storage

One of those cryptocurrencies that offer a utility is Sia Coin. Introduced in 2014, Sia Coin is "a platform for decentralized storage" that "enables the formation of storage contracts between peers." [14]

2016 Golem – Decentralized Supercomputer

Another utility cryptocurrency is Golem which "is the first truly decentralized supercomputer, creating a global market for computing power." [15]

2017 SingularityNET – Decentralized AIs

Late in 2017, SingularityNET was introduced which "is an open-source protocol and collection of smart contracts for a decentralized market of coordinated AI services." [16]

Architecture Overview

Mining cryptocurrencies is not new. In fact, ever since Bitcoin [5] was introduced in 2009, cryptocurrencies have exploded. In 2013, Ethereum [10], added the idea of the 'smart contract' [13] to the cryptocurrency which greatly expanded its use and exploded into numerous of cryptocurrency coins and tokens. Both Bitcoin and Ethereum (for now) are mined using a proof of work algorithm [7] [17]. Mining, is the process of verifying each transaction and a number of software programs are offered to perform this task, such as the Claymore Dual Miner [2].

The Universal Mining System is designed to offer a flexible, modular system that combines the plug-n-play functionality of component models with that of using shared computing power to solve a problem, yet do so in a way that is compensated much in the way that cryptocurrency miners are compensated today. This model allows for adding new miner capabilities after the product is installed, and do so without requiring changes to the overall mining system. For example, once installed, the user can easily add in the future new mining support for a new cryptocurrency not currently supported, merely by installing a new mining module that mines that new cryptocurrency.

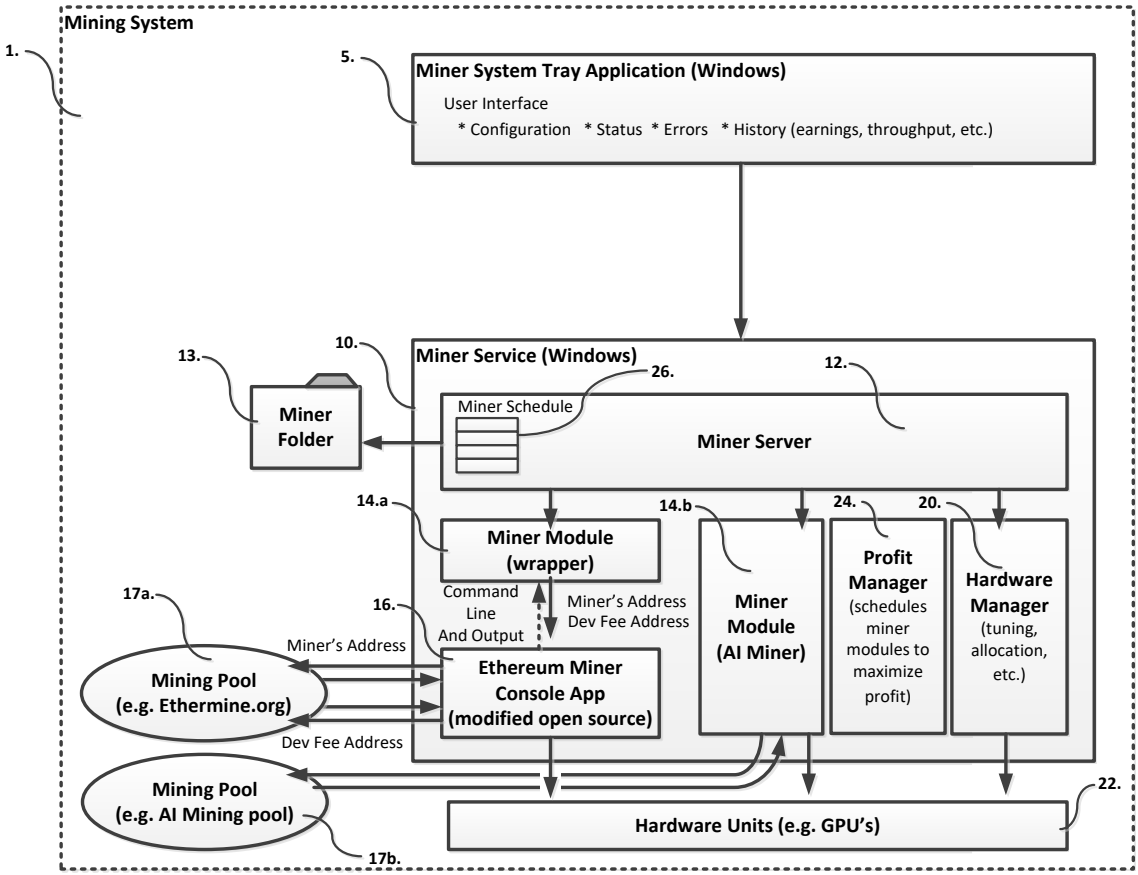


Figure 1 Universal Mining System

The Mining System (1) comprises a Miner System Tray Application (5) used by the user to interact with the Miner Service (10) which in the example above runs as part of the Windows operating system. The communication between the Miner System Tray Application (5) and the Miner Service (10) may use various communication techniques such as a custom TCP/IP protocol, named pipes, shared memory, Web Service Protocols (such as the Windows Communication Foundation or WCF [18] [19]), or REST based API. The Miner Service (10), internally uses the Miner Server (12) to manage the overall mining process with the goal of maximizing the existing system resources to run mining operations. Optionally, a Miner Folder (13) is used by the Miner Server (12) to load the set of Miner Modules (14) that are available at the time. Other methods may also be used to locate, or otherwise discover Miner Modules (14) to use. For example, the Miner Server (12) may query a site on the Internet for Miner Modules (14), download those modules and use them. Alternatively, the Miner Server (12) may query a database for the Miner Modules (14) to use. Or, when running the Mining System (1) or portions of the Mining System (1), such as the Miner Server (12), on a phone, the phone network may be used to query a set of one or more Miner Modules (14) to use. It is important to note that the Miner Server (12) may use just one instance of a Miner Module (14) at a time, or one or more instances of the same Miner Module (14) at the same time where each Miner Module (14) uses a different configuration, or use two or more instances at the same time of different Miner Modules (14) that each perform different mining tasks (or media 'playing' tasks). For example the Miner Server (12) may use an Ethereum Mining Module (14.a) to mine Ethereum while at the same time also use an AI Mining Module (14.b) to solve artificial intelligence problems. Or, alternatively, the Miner Server may use one instance of the Ethereum Mining Module (14.a) that is configured to run on GPU's 0-2, and another instance of the Ethereum Mining Module that is configured to run GPU's 3-4. This latter scenario may be useful in that it would allow for better scheduling in that GPU's 0-2 may be used on a 24/7 basis, whereas the GPU's 3-4 may only be used at night.

Miner Modules (14) can take several forms. For example a Miner Module (14) may actually be a Miner Module Wrapper (14a), whereby the Miner Module Wrapper (14a) delegates the mining commands to the actual Miner, such as an open-source Ethereum Miner Console App (16.). Alternatively, a Miner Module (14) may implement the mining operations itself, such as an AI Miner Module (14b.). Or, the Miner Module (14) may perform a mining task such as mining AI, or performing a search for SETI similar to those performed by BOINC from Berkeley SETI [6], and delegate the AI related tasks to an AI system. Such AI systems may include one running in a local container (such as a Docker Container), or an AI system running in a remote container (such as a Docker Container) in the cloud, or an AI system running on a remote machine in the cloud, or an AI Platform running on the local machine.

Each miner, such as the Ethereum Miner Console App (16) may interact directly with a node managing an Ethereum blockchain, or alternatively, the miner may interact with a Ethereum Mining Pool (17a.) such as the one offered by Ethermine.org. Either way, the

Miner Module such as the Ethereum Miner Console App (16), uses the direct node or pool to query for work and post results which are then compensated for via the pool, or node.

A Miner Module (14) such as the AI Miner Module (14b) may also interact directly with a node within a blockchain from which the AI Miner Module works, or from an AI Mining Pool (17b) to get work, post results, and receive compensation.

Alternatively, an AI Miner Module (14b) can be created to query AI work packages for a server (locally or over the Internet), and send the results back to that server, or another server designated to receive the results, and upon completion, receive compensation in the form of a cryptocurrency (such as Ethereum), PayPal payment, visa transaction, or some other form of compensation.

While Miner Module(s) (14) run, the Miner Server (12) also runs the Hardware Manager (20) which is responsible for tuning, optimizing and allocating the Hardware Units (such as Graphical Processing Units or GPUs) (22) within the system. These Hardware Units (22) may reside on the local machine or may be networked across a vast collection of computers in a local network, or may even be in a remote network such a cloud environment. The main goal of the Hardware Manager (20) is to optimize the use of the hardware resources and make those resources available to the Miner Module(s) (14).

In addition, a Profit Manager (24) may be used as well that acts similar to the Hardware Manager (20), but instead of optimizing the hardware resources, the Profit Manager (24) optimizes which Miner Module(s) (14) to run on the available resources in order to maximize the overall profit of the Mining System (1). By using the current compensation of each mining target (e.g. Ethereum cryptocurrency, AI currency, AI task, etc.) and the overall difficulty to reach that compensation, the Profit Manager (24) dynamically calculates the overall compensation of mining each target and periodically adjusts the schedule for which each Mining Module (14) is run and does so in such a way that maximizes the overall profit.

The Profit Manager (24) queries each Miner Module (14) for the current price and mining difficulty of the mining target mined by the Miner Module (14). Using this information, the Profit Manager monitors trends in the overall profit of each miner and then updates the Miner Module (14) run Miner Schedule (26) managed by the Miner Server (12) or one of its modules. When not using the Profit Manager (14), which may also be implemented as a part of the Miner Server (12), the Miner Server (12) uses a default schedule to run each Mining Module. The Profit Manager (24) may track the overall performance of each Miner Module (14) and use machine learning techniques such as deep neural networks to determine the most optimal schedule for the set of Miner Modules (14) used.

In a common scenario, the Miner Server (12), uses the Hardware Manger (20) to query the available resources, and then uses the Profit Manager (24) to build a schedule that describes which Miner Module(s) (14) will use those resources and when.

USE CASE – INITIALIZATION

When first starting up the Mining System (1), there are a sequence of steps that take place to get the system up and running. Given the main goal of the Mining System to run each Mining Module optimally at all times, it is important that the system start up and run right away with the current set of user configurations.

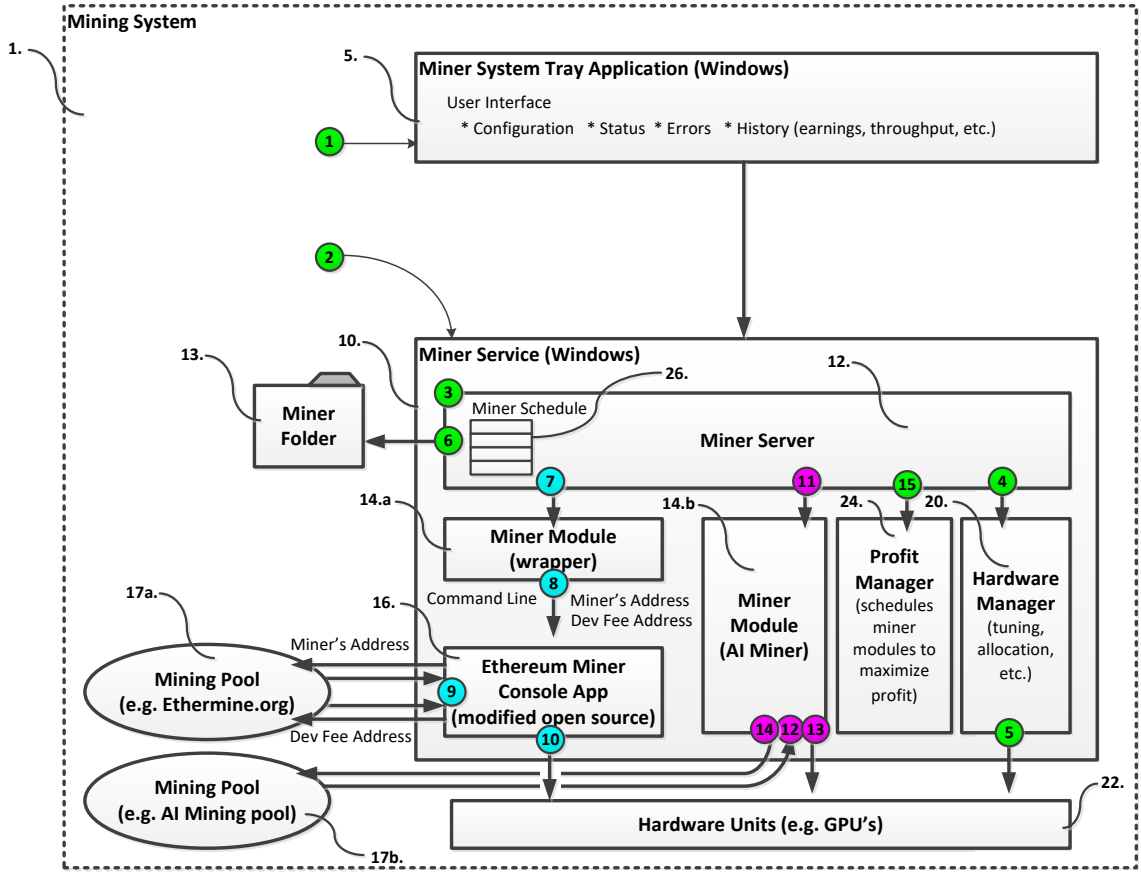


Figure 2 Initializing the Mining System

When starting up the Mining System, the following steps take place.

1.) When the user is logged in, the Miner System Tray Application (5) automatically starts running and is displayed in the lower right hand corner of the exemplary Windows bases system. This may also run as a general application, or a control panel applet, a phone applet, or even a web browser. The main point of this application is to provide the user a way of configuring the system directing it how to run and querying and displaying status to the user (or another automated system) while it runs.

2.) In a common scenario, the operating system automatically starts the Miner Service (10). However, the Miner Service (10) may also be run as an application, a phone app, a background application, Web Service, Azure Web Role, or other type of

application or service. The main task of the Miner Service (10) is to manage running the overall operations provided by the Miner Server (11).

3.) Upon starting, the Miner Service (10) starts the Miner Server (12), for example in a dedicated thread.
4.) The Miner Server (12) then starts the Hardware Manager (20)…
5.) …which queries the current Hardware Units (22) such as the Graphical Processing Units (GPUs) that are available along with their capabilities such as model number, memory, and processor speed, etc.
6.) Next the Miner Server (12), loads all Miner Modules (14), or Miner Module (14) configurations from the Miner Folder (13) and starts each one running based on the Miner Server's (12) default Miner Schedule (26), which allocates each hardware resource to each Miner Module (14) loaded.
7.) When run, the Miner Wrapper Modules (14a), such as the Ethereum Miner Wrapper…
8.) … start their underlying miner, such as the Ethereum Mining Console App (16) run by the Ethereum Miner Wrapper Module.
9.) During operation, the Ethereum Mining Console App (16) runs in the same way as it would when run stand alone in that is begins mining the Ethereum cryptocurrency by either interacting directly with an Ethereum blockchain node or with an Ethereum mining pool (17a.) such as the one run by Ethermine.org.
10.) While mining, the Ethereum Mining Console App (16) mines the Ethereum cryptocurrency using the GPU's allocated to it in step #6 above.
11.) Other full Miner Modules such as the AI Mining Module (14b) also start when scheduled.
12.) When run, the AI Mining Module (14b) queries an AI Mining Pool (17b) or a direct AI blockchain node for work.
13.) Upon receiving the work, the AI Mining Module (14b) begins solving the AI problem using the Hardware Units (22) allocated to it. In a common scenario, the Hardware Units (22) used are Graphical Processing Units (GPU's) such as those sold by NVIDIA or AMD.
14.) Upon solving the AI problem, the solution is sent back to the AI Mining Pool (17b) or direct AI blockchain node for verification. Once verified, the compensation is sent to the blockchain address (or visa card, or PayPal account or other account) specified by the AI Miner Module (14b).
15.) After starting up each Miner Module (14) using the default Miner Schedule (26), the list of Miner Modules is sent to the Profit Manager (24) which in turn queries each module for their compensation metrics, difficulty and other information necessary to determine the overall profit/time ratio of each Miner Module (14). Using this information, the Profit Manager (24) then builds a new Miner Schedule (26) which it sends back to the Miner Server (12) for use during the next mining cycle.

USE CASE – RESCHEDULING

While running, the Miner Server (12) periodically runs the Profit Manager (24) in order to update its Miner Schedule (26). The following use case describes how this is done.

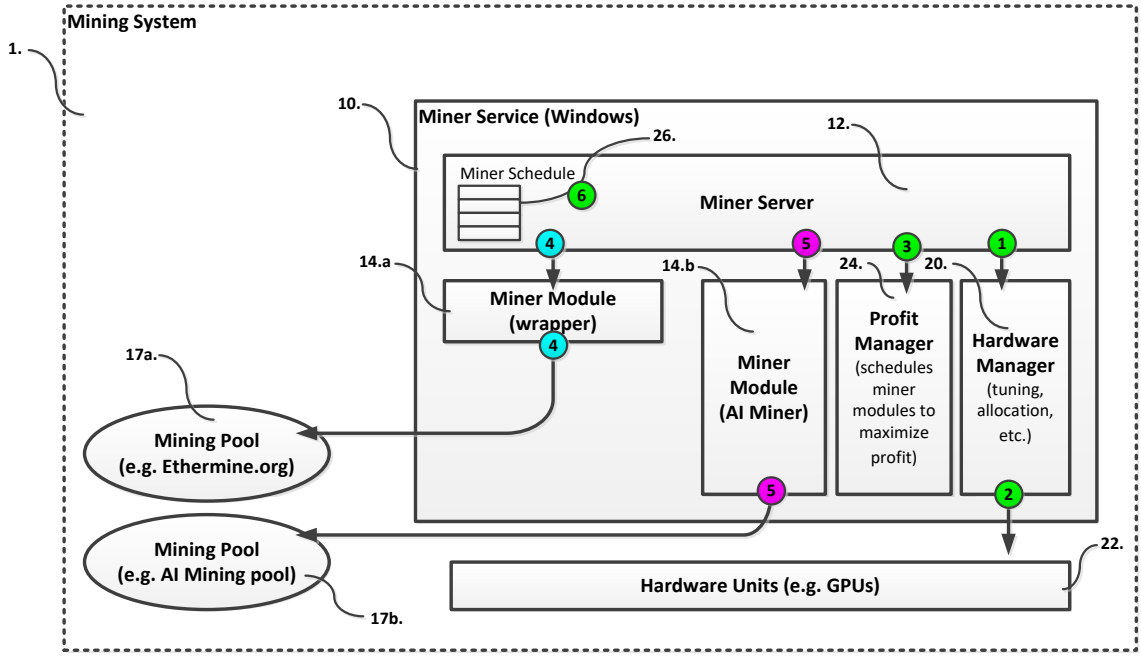


Figure 3 Miner Scheduling

The following steps occur when rescheduling the Miner Modules (14).

1.) First the Miner Server (12) queries the Hardware Manager (20) for the current performance statistics of the processing and memory resources used.
2.) The Hardware Manager continually monitors the Hardware Units (22) used and when queried, returns this information to the Miner Server (12).
3.) Next the, Miner Server (12), passes the list of available Miner Modules (14) and performance statistics to the Profit Manager (24).
4.) The Profit Manager (24) then queries each Miner Module Wrapper (14a) for the underlying compensation and difficulty information, such as current cryptocurrency price, etc.
5.) In addition, the Profit Manager (24) queries each full Miner Module (14b) for its underlying compensation and difficulty information.
6.) Using the information collected, the Profit Manager (24) calculates the optimal Miner Schedule which is then returned to the Miner Server (12) and run upon the completion of the currently running schedule, or starts running immediately if no schedule is currently running.

USE CASE – QUERYING INFORMATION

While running the Mining System (1), it is helpful to give the user running the system (or an automated system running the system) information such as status, profitability and hardware performance data so that they can monitor how well their system is running.

In addition, the user of (or automated system managing) the Mining System (1) need a way to configure the system. For example the user may only want to run the system at certain times of the day, or only on specific GPU hardware resources.

The Miner System Tray Application is an example of an application that allows for monitoring and/or configuring the system. Note, this application may also be a web page viewed and/or interacted with via a web browser, or it may be a phone app, or even an automated scripting environment run by an automated system.

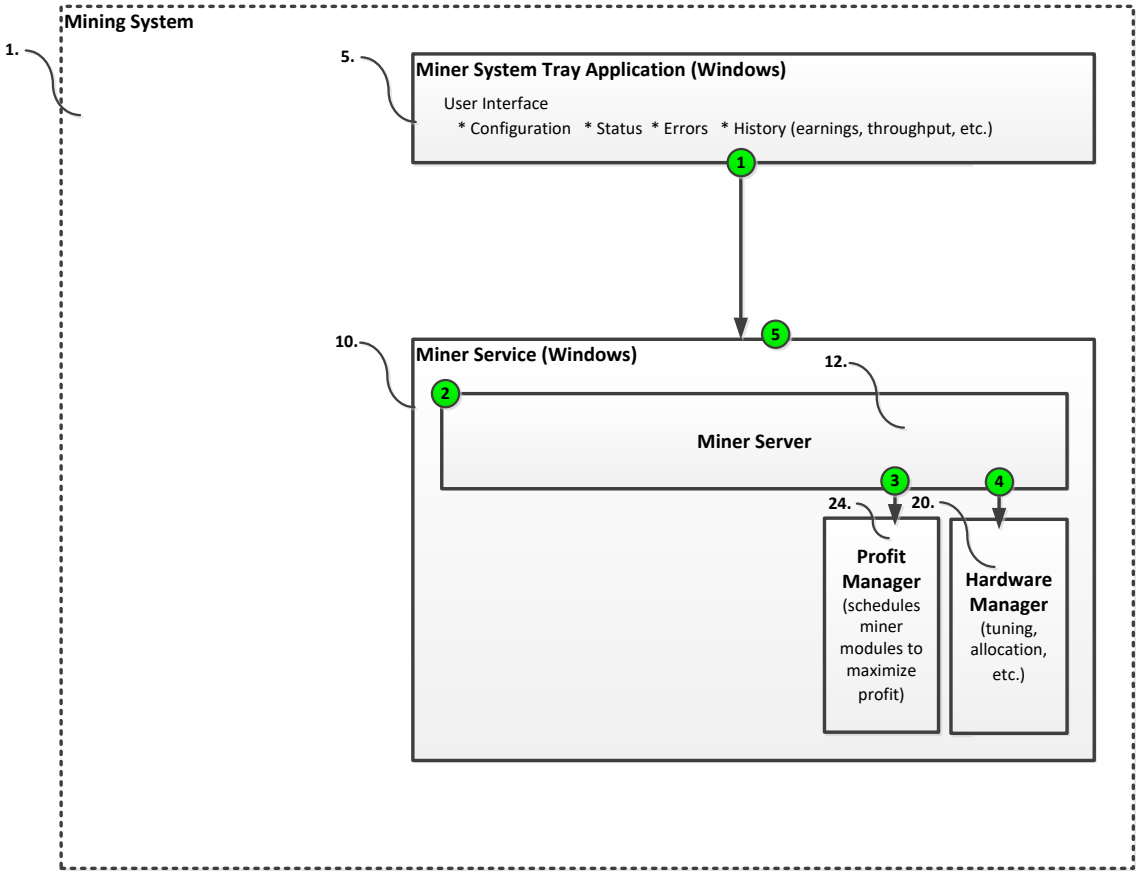


Figure 4 Monitoring and Configuring the Mining System

When monitoring and configuring the Mining System, the following steps occur.

1.) First the application, such as the exemplary Miner System Tray Application, queries the Miner Service (10) for information that it would like to display. For example it may query the Miner Service (10) for its overall profitability over a time period.

2.) The Miner Service (10) sends the query to the Miner Server (12).
3.) The Miner Server (12) queries the Profit Manager (24) for the profitability statistics that it has collected…
4.) …and then the Miner Server (12) queries the Hardware Manager (20) for the hardware performance statistics that it has collected.
5.) The collected information is then sent back to the application and displayed to the user.

Configuration settings are also sent from the application to the Miner Service (10) which then delegates the settings to the Miner Server (12) to configure each of the appropriate modules.

User Interface

In the exemplary system, application used to configure and monitor the Mining Service (10) is a Windows based system tray application. While running, this application displays a small status icon in the lower right hand corner of the windows desktop.

Double clicking on (or right clicking on to display the menu) opens the applications user interface.

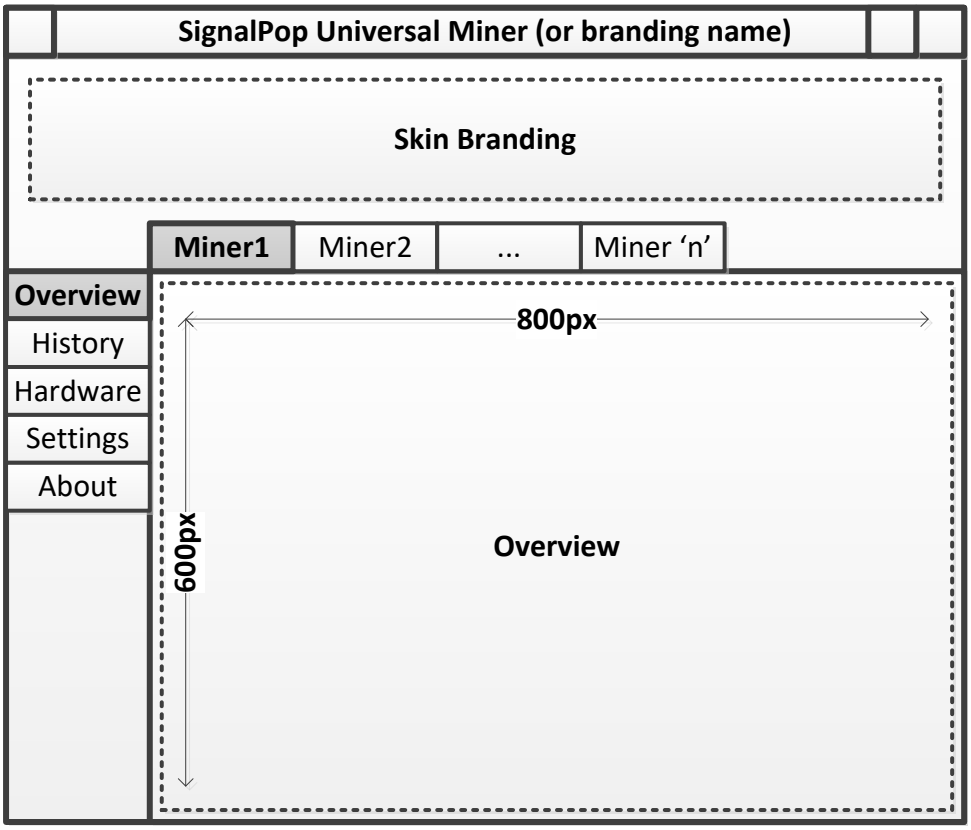


Figure 5 SignalPop Miner Main Window

The exemplary system tray applet displays the main window above when double clicked. This application could also launce as a stand-alone application, run as a phone app or be accessed via a set of web pages viewed through a web browser.

The application user interface is designed to support a 'skin' model where the coloring, branding images, and labeling of items are all interchanged based on a configuration file prepared by an OEM partner, or by SignalPop on behalf of an OEM partner.

Three main areas make up the application: 1.) The skin branding area which allows OEM partners to display logo and other information that rotates on a schedule thus allowing for more information; 2.) The menu-bar area which contains menu items; 3.) and the main

window area which in the above picture shows the 'Statistics' window. The following sections describe each of the main windows that are available.

OVERVIEW

The overview section gives the basic, essential information of each miner module that each miner monitors carefully in order to maximize their throughput, yet do so without overheating. This view is optionally provided by the Miner Module itself. However, in a common scenario, the Miner Module just provides a set of key pairs that contain its key data points and statistics.

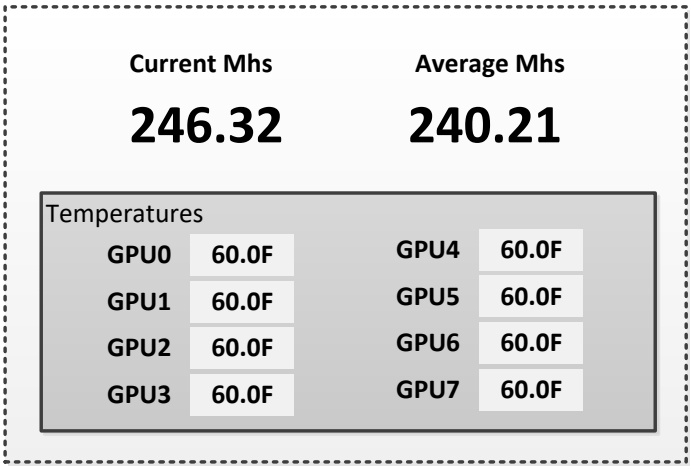


Figure 6 Overview View

HISTORY

The history section shows historical information such as the MHs over time and the GPU temperatures over time.

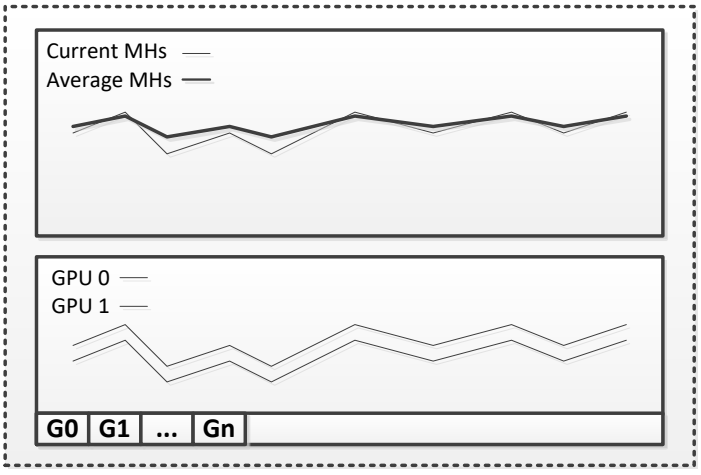


Figure 7 Historical View

HARDWARE

The hardware section gives detailed information on each hardware unit (e.g. GPU) used such as the units temperature, usage, memory, etc.

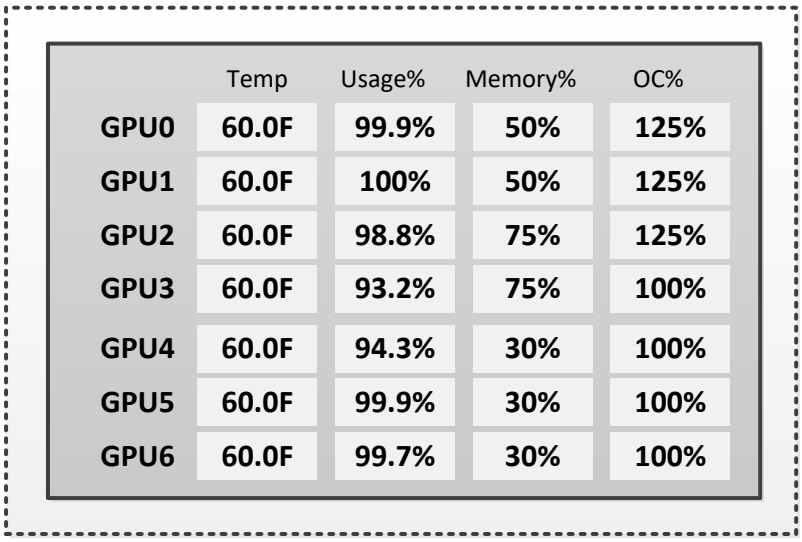


Figure 8 Hardware View

SETTINGS

The settings section is an area that allows the user to configure the underlying miner modules. A 'general' tab allows for setting global settings.

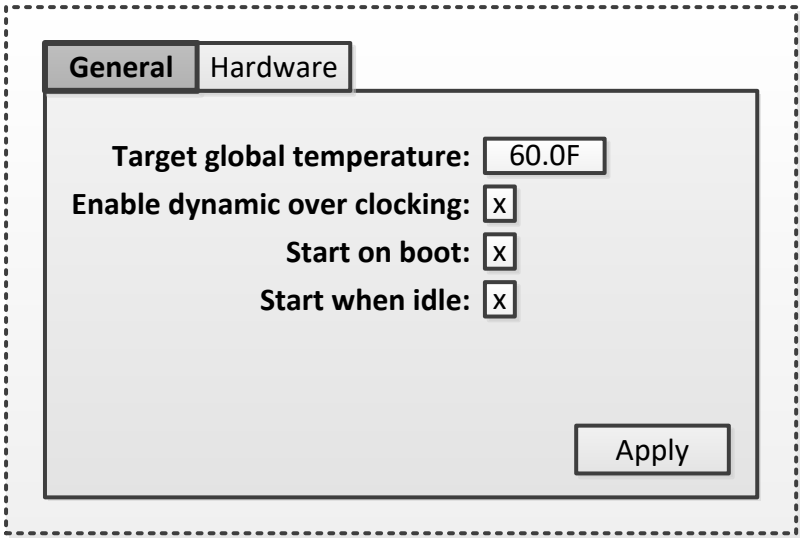
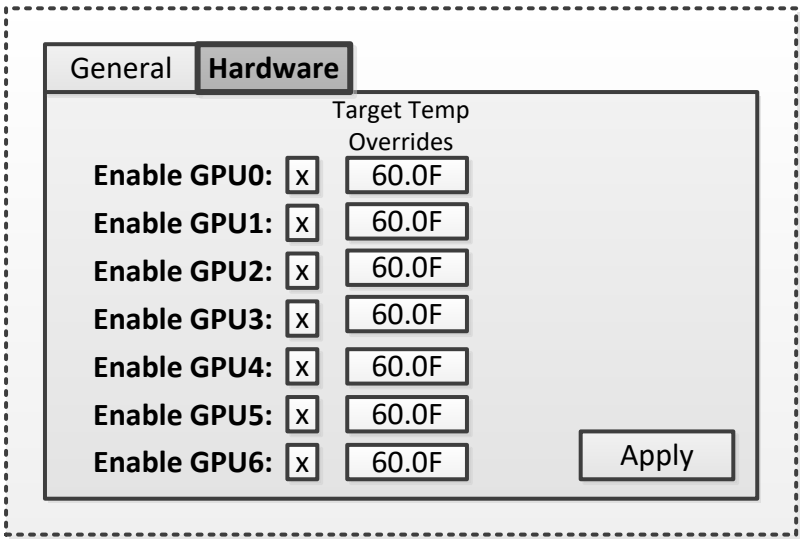


Figure 9 General Settings View

And the 'hardware' tab allows for setting hardware related settings.



ABOUT

The about section is an area that allows for more detailed OEM information and branding.

Figure 10 About View

API/SPI Model

Similar to the Java Naming and Directory Interface (which has a JNDI API and JNDI SPI) discussed in Section 4.3 of [4], the Universal Miner System also uses an API and SPI each of which are specific to the task of mining.

The Mining Server (12) exposes the Mining API which is then used by the Mining System Tray Application (5). And each Miner Module (14) exposes the Mining SPI which is then used by the Mining Server (12).

Note since the Mining Service (10) is a simple container for the Mining Server (12) it either passes along the API to the software users that call the API, or alternatively, the API is exposed via a communication protocol such as WFC [19] [18] or a REST based protocol. The Miner SPI could also use such protocols in a distributed system where a Miner Module (14) resides on a different machine, or runs in a different process (on the same machine) than the Miner Server (12). In closely connected systems, the SPI and API may also be implemented as a more traditional set of functions that are called directly by the software that uses the functions.

The following sections describe the functions that make up both the Miner API and Miner SPI set of functions.

COMMON TYPES

The following types are used by both the API and SPI functions.

Parameter Type

The *Parameter* type is used to store a single parameter sent to a Run or Query function.

```
struct Parameter // pseudo code
{
   string    m_strName;
   double?   m_dfVal;
   string    m_strVal;
   byte[]    m_rgBytes;
   Exception m_error;
   TYPE      m_type;

   enum TYPE { NUMERIC, STRING, BYTES, ERROR }
}
```


Result Type

The *Result* type is used to store a single result value returned by a Run or Command function.

```
struct Result // pseudo code
{
  Parameter[] m_values;
  long        m_lcode;
}
```

COMMAND Type

The COMMAND type defines the commands to run. The following are exemplary commands.

```
enum COMMAND
{
   INITIALIZE,
   RUN,
   STOP
}
```

QUERY Type

The QUERY type defines the information to be queried. The following are exemplary queries.

```
enum QUERY
{
   MINER_MODULES,
   STATUS,
   SCHEDULE
}
```


MINER API

The following exemplary functions make up the Miner Application Programming Interface (Miner API) that is implemented by the Miner Server (12) or one of its modules.

Run API

The Run API is used to run commands that cause the Miner Server (12) to do something such as initialize, configure, start, stop, etc.

Syntax	`Result Run(COMMAND cmd, Parameters[] rgP)`
Parameters	*COMMAND* **cmd** – specifies the command to run. *Parameter[]* **rgP** – specifies an array of parameters.
Exceptions	An exception is thrown on error.
Returns	The results (if any) are returned as a *Result* value.

Query API

The Query QPI is used to query information from the Miner Server (12) such as the current miner modules that are running, or the status of each module.

Syntax	`Result Query(QUERY qry, Parameters[] rgP)`
Parameters	*QUERY* **qry** – specifies the command to run. *Parameter[]* **rgP** – specifies an array of parameters.
Exceptions	An exception is thrown on error.
Returns	The results are returned as a *Result* value.

MINER SPI

The following exemplary functions make up the Miner Service Provider Interface (Miner SPI) and are implemented by each of the Miner Module (14) or one of their modules.

Run SPI

The Run SPI is used to run commands that cause the Miner Module (14) to do something such as initialize, configure, start, stop, etc. Note, the example below shows the SPI supporting the same COMMAND's as the API. However, the commands supported by the SPI may be entirely different that the commands supported by the API.

Syntax	`Result Run(COMMAND cmd, Parameters[] rgP)`
Parameters	*COMMAND* **cmd** – specifies the command to run. *Parameter[]* **rgP** – specifies an array of parameters.
Exceptions	An exception is thrown on error.
Returns	The results (if any) are returned as a *Result* value.

Query API

The Query QPI is used to query information from the Miner Server (12) such as the current miner modules that are running, or the status of each module. Note, the example below shows the SPI supporting the same QUERY's as the API. However, the queries supported by the SPI may be entirely different that the queries supported by the API.

Syntax	`Result Query(QUERY qry, Parameters[] rgP)`
Parameters	*QUERY* **qry** – specifies the command to run. *Parameter[]* **rgP** – specifies an array of parameters.
Exceptions	An exception is thrown on error.
Returns	The results are returned as a *Result* value.

Additional Use Cases

Up until now, we have discussed a miner that solves a computational problem. In addition to these uses, the Mining System (1) can also be used to perform operations on media such as music or video. For example, a Mining Module (14) may be used to verify the ownership, content (e.g. parental controls), or licensing (media rights management) that allow (or disallow) a user to play media content.

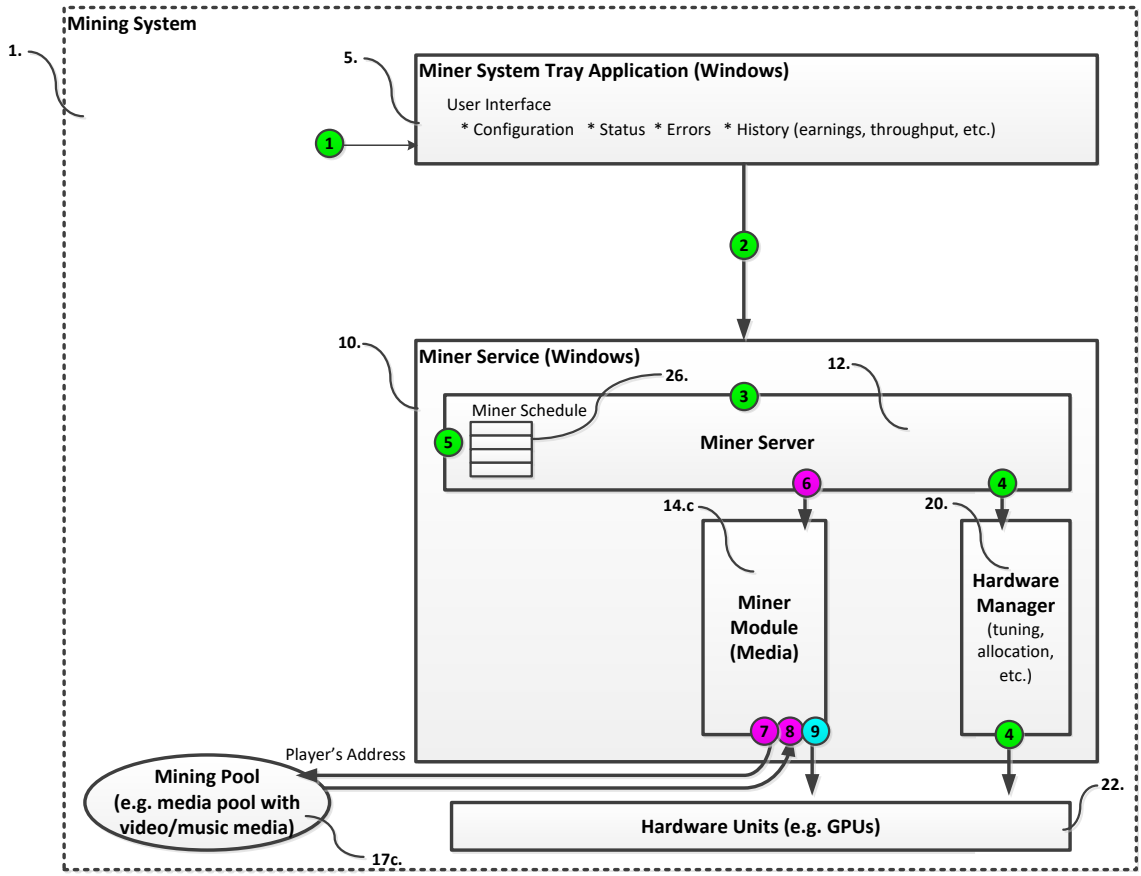


Figure 11 Mining System for Media

When using the Mining System (1) to either verify whether or not content can be played, or to actually play the content by viewing the media (movie) or listening to the media (music), the following steps take place.

1.) First the user uses the exemplary Media System Tray Application (5) to cause the Media Service (10) to either verify whether content can be played, plays the media, or both.
2.) Next, the Media Service (10) passes the request to the Miner Server (12). Alternatively, if the Miner Server (12) directly implements an API interface via a protocol such as WFC [18], then the Media System Tray Application's (5) request goes directly to the Miner Server (12).

3.) The Miner Server (12) then processes the request by…
4.) … querying the Hardware Manager (20) for available hardware resources used to either verify or play the media (or both). In the event that a Hardware Manager (20) is not used, the Miner Server (12) may just query the operating system or drivers installed within the operating system for this information.
5.) Next, the Miner Server (12) may optionally use the Miner Schedule (26) to see when to play the media in the event that a media play request was received. For example, the Miner Schedule may define when to play certain songs at certain times of the day.
6.) Next, the Miner Server (12) directs the Miner Module for Media (14.c) to play (or verify for playing) the media.
7.) The Miner Module for Media (14.c) then queries a Mining Pool for Media (17.c), which in this case is actually a media pool containing media such as videos, music, art, games, gaming content, etc. When querying the media, the Miner Module for Media (17.c) passes along the user's 'Player Address' which is an address in the blockchain supported by the media pool. Alternatively, the Miner Module for Media (14.c) may use similar steps to get the content from a direct node in the media based blockchain.
8.) Next, the Miner Module for Media either receives the content, or in the case that it is verifying its ability to play the media, receives a confirmation (or objection) to play the media.
9.) In the case that the Miner Module for Media is playing the content, if it receives a confirmation that it can play the media, the Miner Module begins playing the media on the hardware allocated to it. Alternatively, the Miner Module for Media may delegate the playing functionality to another software system such as a media player (e.g. movie player, music player, game, gaming system, etc.).

As shown, the Mining System (1) not only performs tasks that use computational power to solve a computational problem, it can also use those same computational powers to 'play' media.

Exemplary media used by the Mining System (1) include: video, music, gaming content, full games, gaming modules, gaming characters, web content, artistic renderings, art, 3d renderings, ray-trace models, 3d rendering models, engineering models, books, book media (automatically read books), research papers, path plans for a robot, driving directions, search results, data files, etc.

Summary

In summary, the Mining System (1) offers, a flexible system that allows solving numerous problems by optimally using a set of computational resources such GPU's. Whether mining a cryptocurrency, solving an artificial intelligence problem, or searching for a solution, the Mining System (1) offers an extremely flexible platform to get the job done.

References

[1] C. Pacia, "Bitcoin Mining Explained Like You're Five: Part 1 - Incentives," Escape Velocity Blog, 2 September 2013. [Online]. Available: https://chrispacia.wordpress.com/2013/09/02/bitcoin-mining-explained-like-youre-five-part-1-incentives/.

[2] Claymore, "Claymore's Dual Ethereum AMD+NVIDIA GPU Miner v11.5 (Windows/Linux)," 11 April 2016. [Online]. Available: https://bitcointalk.org/index.php?topic=1433925.0.

[3] S. Scoles, "A Brief History of SETI&Home," *The Atlantic,* p. 9, May 23, 2017.

[4] R. C. Seacord and L. Wrage, "Replaceable Components and the Service Provider Interface," *Carnegie Mellon University Digital Library,* pp. All pages including 1-27, July 2002.

[5] Wikipedia, "Bitcoin," 2009. [Online]. Available: https://en.wikipedia.org/wiki/Bitcoin.

[6] Berkeley SETI, "Participate - Download BOINC and join the search," BSRC, [Online]. Available: https://seti.berkeley.edu/participate/.

[7] C. Pacia, "Bitcoin Mining Explained Like You're Five: Part 2 - Mechanics," Escape Velocity Blog, 2 September 2013. [Online]. Available: https://chrispacia.wordpress.com/2013/09/02/bitcoin-mining-explained-like-youre-five-part-2-mechanics/.

[8] C. Pacia, "Bitcoin Explained Like You're Five: Part 3 - Cryptography," Escape Velocity Blog, 7 September 2013. [Online]. Available: https://chrispacia.wordpress.com/2013/09/07/bitcoin-cryptography-digital-signatures-explained/.

[9] C. Pacia, "Bitcoin Explained Like You're Five: Part 4 - Securing Your Wallet," Escape Velocity Blog, 29 September 2013. [Online]. Available: https://chrispacia.wordpress.com/2013/09/29/bitcoin-explained-like-youre-five-part-4-securing-your-wallet/.

[10] Wikipedia, "Ethereum," 2013. [Online]. Available: https://en.wikipedia.org/wiki/Ethereum.

[11] N. Szabo, "Smart Contracts: Building Blocks for Digital Markets," 1996. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.

[12] J. W. Investor, "Quora," 29 August 2017. [Online]. Available: https://www.quora.com/What-is-the-difference-between-SiaCoin-Golem-and-Ethereum.

[13] P. Humiston, "Build Your First Ethereum Smart Contract with Solidity - Tutorial," codeburst.io, 12 November 2017. [Online]. Available: https://codeburst.io/build-your-first-ethereum-smart-contract-with-solidity-tutorial-94171d6b1c4b.

[14] D. Vorick and L. Champine, "Sia: Simple Decentralized Storage," 2014. [Online]. Available: https://sia.tech/sia.pdf.

[15] Golem, "The Golemm Project," 2016, October. [Online]. Available: http://golemproject.net/doc/DraftGolemProjectWhitepaper.pdf.

[16] SingularityNET, "SingularityNET: A decentralized, open market and inter-network for AIs," 19 December 2017. [Online]. Available: https://public.singularitynet.io/whitepaper.pdf.

[17] M. J. Casey and P. Vigna, "In blockchain we trust," *MIT Technology Review: Blockchain,* pp. 10-16, 9 April 2018.

[18] Wikipedia, "Windows Communication Foundation," Wikipedia, Est. 2006. [Online]. Available: https://en.wikipedia.org/wiki/Windows_Communication_Foundation.

[19] B. Hollunder, "Code Contracts for Windows Communication Foundation (WCF)," *Service Computation 2010: The Second International Conferences on Advanced Service Computing,* no. ISBN: 978-1-61208-105-2, pp. 14-20, 2010.

