AI Network Architecture Paper

A blockchain designed to manage P2P computation cloud

Last updated on June 29th, 2019

Contents

1 Introduction **1**
1.1 Vision 1
1.2 Background 2

2 System Architecture **3**
2.1 Deep Computing Architecture 3
2.1.1 Overview 3
2.1.2 History 3
2.1.3 Technical Challenges 4
2.1.4 Execution Correctness and Solution Correctness 5
2.2 Solution 6
2.2.1 Secure Runtime Environment 7
2.2.2 Big Storage 8
2.2.3 AIN Blockchain Protocol 8
2.2.4 Machine Learning Patterns on AIN protocol 18
2.2.5 Security 21

3 Sample User Experience **23**

1 Introduction

1.1 Vision

Build a global computer network where anyone can provide and compose cutting edge solutions with open access to shared computing resources in a cost-effective way.

1.2 Background

There is a plethora of cloud-based managed computing platforms operated by large companies such as Amazon, Google, IBM, Microsoft, etc. By leveraging economies of scale via centralization, these companies provide cost-effective computing solutions for enterprises. However, the rigidity of such centralized resource management serve as a barrier to entry for individual developers and small-scale research groups. The following are the three major challenges of AI development in a resource constrained environment.

First, large-scale Machine Learning (ML) problems are often expensive and requires complex settings. When cutting-edge ML research papers are published, results are often difficult to replicate. Even when results are replicable, it can sometimes takes a very long time to train the models. In our platform, runtime environments are shared together with code so that anyone can efficiently access and improve ML solutions.

Second, private clouds established by small research groups or companies do not operate cost-effectively. In addition to research, private clouds are needed for running ML jobs across a diverse range of domains. Once a private cloud is set up, it's often the case that only a small number of researchers use this cloud at any given time. This means cloud resources are not always fully utilized. We would like to provide ways to save idle computing resources in a form that they can be utilized later when it's needed. For example, if one lends its computer power during the night, it can run 2x computer power during the day.

Third, big AI companies often reserve their most powerful resources primarily for themselves first. For example, these companies may make new hardware accessible only to privately selected customers before general release. In such cases, small customers may have no other option than to accept the vendor's policy even though it doesn't fit well to the customers' needs. We propose a neutral market where innovative solutions can be run and sahed independently from central providers.
We believe that blockchain can provide this neutral market for sharing computing resources without the limitations of centralized vendors. Here are some of the benefits that different participants of blockchain network can gain:

1. *Miners can run jobs which may be more profitable than mining.*
2. *Researchers can easily access state-of-the-art solutions and run them effortlessly.*
3. *Server Farms can reduce resource idle time and maximize the performance during active time.*

By exploiting the key properties of blockchain such as transparency and decentralization, we want to make a paradigm shift from dedicating massive amounts of computing resources to mining (i.e., running identical hash operations in parallel) to building a federation of distributed computing resources that solves real-world computing problems.

2 System Architecture

2.1 Deep Computing Architecture

2.1.1 Overview

In this section, we will outline the technical details of Deep Computing Architecture. This architecture is a distributed P2P cloud that can solve large heterogeneous problems with minimal centralized control. At the center of the blockchain protocol, AIN cryptocurrency makes massive-scale computing accessible by allowing participants to spread and trade computational jobs amongst themselves. We take a hybrid approach to decentralization, combining the inherent decentralized characteristics of blockchain with centralized helper components (Secure Runtime Environment and Big Storage) for trustful distributed computing of high performance. We will discuss how to identify and trust the computation power of distributed machines with disjoint ownerships so that they can be assigned with proper jobs and rewarded with AIN.

2.1.2 History

Ethereum, a blockchain app platform, may be the most popular solution for solving centralized computing problems to date. Ethereum is often considered to be a "World Computer", in the sense that it runs smart contracts which no central entity can shut down. Ethereum also provides the Turing-complete virtual machine. This in principle allows any computational problem, including ML problems, to be solved through Ethereum. While we honor the value of pure P2P systems, we have a slightly different opinion about Etherium's roadmap roadmap for a "World Computer."

In the vision of a fully decentralized web by Taylor Gerring, the co-founder of the Ethereum Foundation, three key components are mentioned: Contracts (decentralized logic), Swarm (decentralized storage), and Whisper (decentralized messaging).

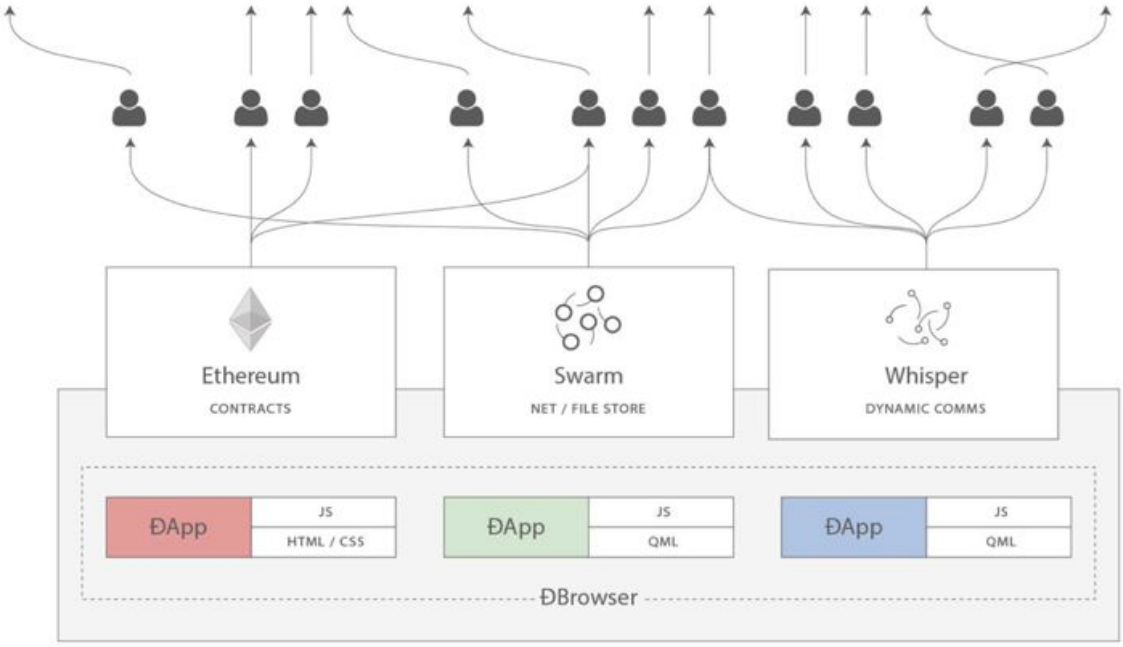

Figure 1. Interaction including Ethereum contracts, Swarm storage, Whisper comms

However, even the most advanced decentralized storage is much more expensive and slower than centralized storage. Furthermore, Ethereum smart contracts are too expensive and not expressive enough to represent complex programs such as ML algorithms.
Practically, in a performance critical area, we believe that a decentralized ecosystem should be able to embrace the efficiency of centralized entities. We acknowledge that a blockchain can only contain small-sized information, so we let the blockchain take responsibility for establishing trust between entities while letting trusted protocols provide the other necessary services..

2.1.3 Technical Challenges

While the present proposed architecture could be applied to a wide variety of computing problems, we are initially focusing on the use of AIN for ML to explain the architecture with. Sharing computing resources in a P2P network is much more complicated than in a centralized system, as the former should be capable of dealing with various devices, different OS environments, unstable connection between peers, etc. By constraining the problem scope to ML, we distill the main challenges for AIN to the following:

1. Heterogeneous Job Assignment. Unlike homogeneous hash computations, we need to be able to schedule and assign the requested jobs to appropriate computing nodes, since ML jobs demand various combinations of memory, storage size and computational hardware such as GPUs. How can we manage complex cloud computing environment such as TensorFlow Cloud by assigning jobs to trustful computers at reasonable operational costs?
2. Trust and Security. The AIN network needs a way of verifying that the provided codes are harmless to the computation participants. Results of computational jobs also must

be evaluated to confirm that they are correct. In addition, the system needs to provide a scheme to distribute the profits to the honest participants of a problem.

3. Computing and Network Performance. Due to the massive computation requirement and large data size of ML jobs, P2P cloud used to be an unpopular choice for distributed ML computing. Moreover, additional verification layers for trust and security can cause significant performance loss compared to a centralized system, so there should be a mechanism to minimize the verification overhead.

2.1.4 Execution Correctness and Solution Correctness

The main difference between Ethereum and Deep Computing lies in how computer resources are allocated for work and evaluation. Ethereum uses Ethereum Virtual Machine (EVM) to ensure the provided code is run correctly and sequentially. However, verifying every line of code with every computer in the blockchain network becomes inefficient and expensive when we apply the same approach to large applications. For instance, a large-scale optimization problem such as a neural network in deep learning might consist of weights, parameters, and coefficients which are not only great in size, but also dynamically change within a few iterations.

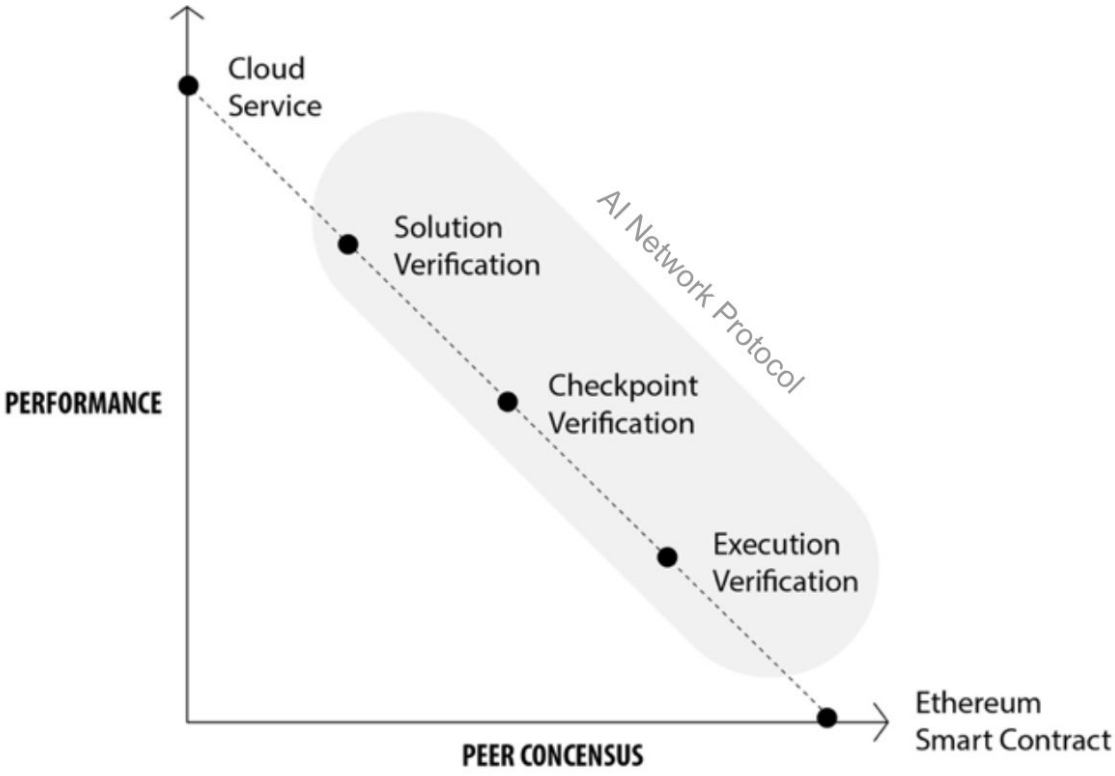

Figure 2. The graph shows performance limitation as operations require more peer consensus

We are normally not interested in the values for this meta-data during interim iterations. Rather, we want to see whether the final iteration produces a solution that satisfies the objective

function for the problem. In addition, processes of solving optimization problems usually includes randomness which might not return predictable results. In this case, it might be sufficient to track whether the model is approaching a satisfactory solution along some expected performance curve.

For performance reasons, we adopted a worker / evaluator model where evaluators verify and rank solutions returned by many workers. Evaluation methods can vary depending on the problem characteristics. Some problems could be satisfied with a simple process that checks whether a solution meets the specified boundaries, while others could utilize a competitive process where evaluators accept the smallest loss in problem test data. Through customized evaluation and verification functions, jobs can consume minimal amount of resources by verifying what it needs. We roughly divided possible verification patten into three categories: solution verification, checkpoint verification and execution verification.

1. Solution Verification. Although finding a solution for a NP problem may be extremely computationally expensive, verifying the solution may be doable in polynomial time.
2. Checkpoint Verification. In order to monitor whether training operations are running as expected, multiple checkpoints during the execution need to be verified. The increased number of checkpoints can provide more solution credibility, at the cost of performance.
3. Execution Verification. One may want to make sure all the lines of codes are executed properly. One obvious and easy way could be executing the same code again across multiple machines. If results are replicable across different machines, this increases credibility. Alternatively, more sophisticated techniques can be used for verifying the code itself. However this verification normally takes more resources than solution verification or checkpoint verification.

2.2 Solution

Our proposed "Deep Computing" solution is a decentralized computational platform that turns GPUs in the network into a global problem solver. Traditional Von Neumann Architectures require computers to contain a CPU, Memory, I/O, and communication between these components. Similarly, we identified three key components we deem essential for a Deep Computer: a secure runtime environment (centralized resource isolation and code verification), big storage service (decentralized storage system [doesn’t need to be fully P2P]), and blockchain (decentralized RPC ledger for reputation control of machines).

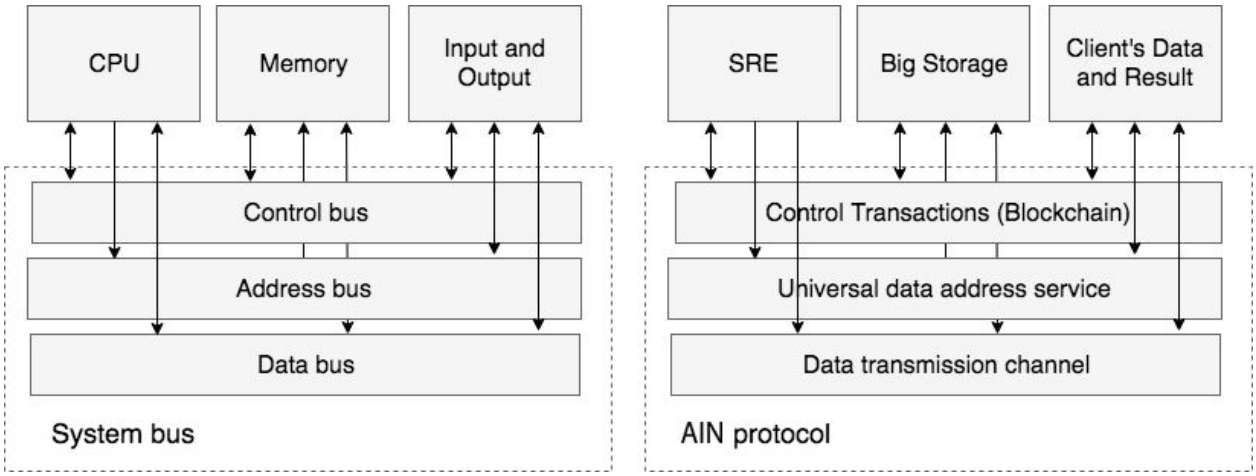

Figure 3. Left shows Von Neumann Architecture and right shows corresponding Deep Computing Architecture. Deep Computing including AIN protocol, Big Storage, Secure Runtime Environment (SRE)

Blockchain defines communication protocol between nodes while tracking every public request between nodes. Processing this information can provide a measurement for reliability of each node, which we call reputation. It also gives reasonable control to centralized nodes and maintains the performance and scalability of distributed computing. With this hybrid approach, we can build a decentralized P2P cloud computation platform which is able to solve various-sized problems in a reliable and efficient way. In the following section, we will describe three main components: Secure Runtime Environment, Big Storage, AIN Blockchain Protocol.

2.2.1 Secure Runtime Environment

To secure user environment and run code on various environments, one needs to provide dynamic sandboxing and enforce resource constraints. The Backend.AI team is the first reference node that will provide a hassle-free backend for AI programming and services. It runs arbitrary user code safely in resource-constrained environments, using Docker and its own sandbox wrapper.

Backend.AI provides a streamlined backend API server that hosts heterogeneous programming languages and popular AI frameworks. It manages the underlying computing resources for multi-tenant computation sessions where such sessions are spawned and executed instantly on demand.

Backend.AI supports various programming languages and runtimes, such as Python 2/3, R, PHP, C/C++, Java, Javascript, Julia, Octave, Haskell, Lua and NodeJS, as well as AI-oriented libraries such as TensorFlow, Keras, Caffe, and MXNet. The implementation details can be found in *https://github.com/lablup/backend.ai* and more frameworks and languages will be added in parallel with this project.

Secure Runtime Environment contains Binary Manager whose role is to authenticate, deploy, and update binaries once they are distributed to the network. When new binaries are released by an admin, Binary Manager notifies the evaluators and the workers to download the new binaries from the Secure Runtime Environment.

2.2.2 Big Storage

Usually an ML job requires problem solving of large-scale input / output data. Furthermore, ML jobs run through decentralization requires efficient data transfer between the network nodes. As such, blockchain itself is not a good solution for storing ML job data, as the blocks are designed to have a limited size for efficiency. Instead, AI Network team will provide a general storage service, what we call Big Storage, that can be used for any type of data sharing or transfer between the nodes. For example, the client can share the input data with the workers and the evaluators, and the workers can share the output data with the client.

For performance reasons, Big Storage is a centralized service that aggregates and allocates different storage services that service providers offer. However, its name service is designed to be universally visible from the blockchain API so that the API allows call-by-reference with only the data path in the storage space. Every node on the blockchain network needs to reserve appropriate space in Big Storage by paying fees to the storage service provider prior to the initiation of an ML job. For instance, an admin will reserve a space for the job scheme, a client will reserve space for the training and test data, and an evaluator and a worker may need to reserve temporary space during the job execution. The size of the available space in the storage space owned by the workers can be referenced by admins for job-worker matching, as some jobs require a certain amount of available storage for the output data.

Basically Big Storage will serve content-addressed storage, i.e., the address is generated by applying a cryptographic hash function (e.g. SHA-2) to the uploaded content. This approach is good as 1) the shared data doesn't need to be updated once it's uploaded in most cases, and 2) no context on the data (including the owner) needs to be exposed in the address.

For the clients who want to encrypt the uploaded data, Big Storage and Blockchain Protocol will provide appropriate encryption and decryption functionalities as well as secure ways to share the decryption keys only with those involved in solving the problem.

When files are created in Big Storage, the file owners can specify the ttl (time to live) of the created files so that they can be automatically deleted after a specified period. This will facilitate efficient management of storage utilization.

2.2.3 AIN Blockchain Protocol

1) Terms

Role

While each node is responsible for maintaining and validating blocks, nodes can also play certain roles for registering and executing other jobs. There are four types of roles in the AIN blockchain: admin, evaluator, worker, and client. The node who registers a new job schema is the root node, and as such is always an admin node.

- Admins can accept other admins, evaluators, or workers. Admins can also hold the roles of worker and evaluator.
- Evaluators validate solutions returned by workers, and assesses the workers' performance. Workers might receive different rewards depending on the results of the assessment and the reward function given by the client.
- Workers execute operations and return solutions.
- Clients request job execution to evaluators and workers, and offer rewards.

Depending on delegation policy, Jobs can be centralized where only one root admin can validate the solution, or scales faster by replicating admins in many places who can accept more workers.

Job Schema

An admin defines a job schema in proto3, and submits its implementation to the binary manager. This schema will eventually be run by elevators and workers. The deployment of the binaries are done by the binary manager, as specified by the owner of the job schema.

Reward

An evaluator validates the results of the workers and assesses the worker's performance as a whole. Rewards are defined through a "reward function" and associated function parameters in the job request. For instance, a reward function may be defined as evenly distributed or as winner-only, depending on the types of problems and the client's preference.

The client needs to prepare a proper strategy when choosing a reward function and function parameters to optimize the cost. AI Network team will provide a few example reward strategies for some of the common use cases.

Deposit

When an evaluator accepts an offer from a client, the evaluator is required to deposit a certain amount of credit. The deposit will be sent to the client if the client can prove the evaluation result was not correct. Deposit must be larger than the fees needed for challenging the evaluation result to the parent node. The details of the challenge process will be explained in a subsequent section.

Fee

Transaction fee exists to prevent from sending a lot of transactions in the purpose of failing the network without any cost. Since AI Network transactions are requests and responses for general purpose application, bandwidth allocation makes more sense than charging a fee for

each transaction. Holding tokens give the node to perform operations in the network, and a job can allocate some tokens to secure bandwidth it requires.

2) Reputation in Blockchain

The blockchain manages ledgers for communication between nodes. This communication includes registration of nodes, offering jobs, accepting jobs, returning results of jobs, evaluating the results, and distributing rewards. Blockchain maintains the specification of API, authentication and ranking of the nodes, offers and acceptances of the jobs, and reward distribution. By tracking this information, we can measure the reputation of each node. Node structure is hierarchical, where the root node has the highest authority to say the solution is correct. Thus, we call it the Reputation Tree (RTree).

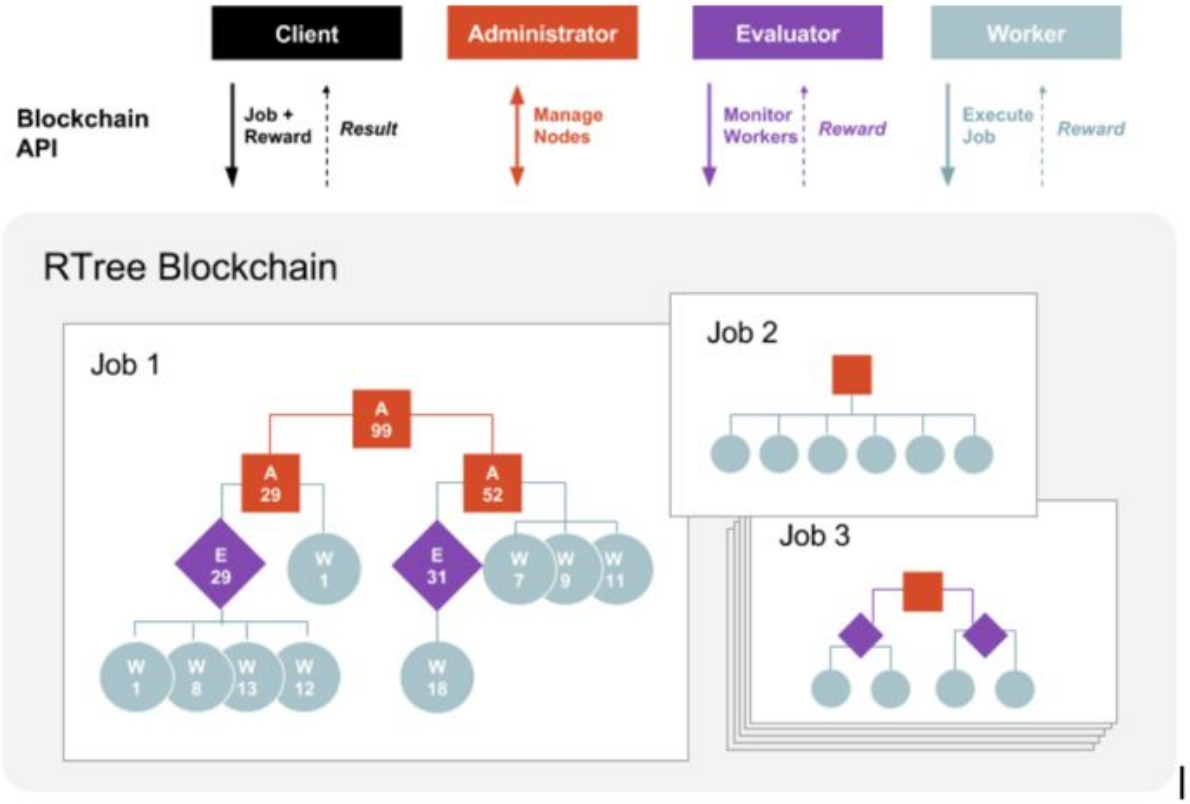

Figure 4. Blockchain can process multiple jobs, and users provide rewards to evaluator and worker for the provided result. Depending on the admin's node expansion policy, it can have multiple admins or multiple evaluators

3) Blockchain API

assignAddress

A new address is assigned to the node. This is similar to creating a wallet in a typical blockchain. Public Key and Secret Key are generated. No Job can be run or managed yet.

registerJob signed by admin

Register a new Job specification. The unique job ID will be created.

PARAMETER	
job_specification (proto)	The information about data, evaluator, and worker service.
RETURNS	
job_id(string)	Unique job ID.

applyJob signed by node
A node applies for a job. The node can apply for the role of either admin, evaluator, and worker.

approveNode signed by admin
When a node applies for a job, an admin node evaluates whether it can truly run the provided functionality using test data. If the admin is satisfied, the node is added as a child of the admin node.

requestJob signed by client
A client sends a job request to the blockchain network. A list of rewards is locked in the blockchain until the final evaluation is produced. If job_input is large, complementary data storage can be provided by AI Network team. If client specifically trusts certain evaluator or worker nodes, the client can specify these nodes.

PARAMETER	
job_id (string)	A unique id of the job to be executed.
job_input (serialized_proto)	The input data to the job. It may contain pointers to big storage.
reward (Function)	A reward function that will be given to the ranked worker
time_limit	An expected duration of job execution, in milliseconds, after an evaluator accepts the job.
evaluator_node (list) [optional]	Candidates that are allowed to participate in the evaluation process.

worker_node (list) [optional]	Candidates that are allowed to participate in executing the job.
skip_evaluation (boolean) [optional]	When a client completely trusts a worker node, the client may skip evaluation. The default value is false.
RETURNS	
transaction_id(string)	Unique ID for created transaction.

acceptJob signed by worker/evaluator
Workers or evaluators can accept a job request if and only if they meet the criteria of the job. The data of the job request can be encrypted and ACL (Access-Control List) controlled. This will make the data only visible to the workers who accepted the job. The client may reject a worker if it doesn't like the worker's reputation.
Workers can also take reputation of other workers into account when deciding whether to execute a job or not. For example, if a job gives the reward to only one worker who produced the best result, other workers might resign if highly competitive workers have already accepted the job.

submitResult signed by worker
Workers produce results and output these results onto the blockchain. The result might be recorded in the complimentary storage if it is too large. The storage provides data integrity at solution submission time.

rank signed by evaluator
After the job time limit has expired, the evaluator sorts the result and assigns a rank to each worker.

acceptResult signed by client
If the client is satisfied with the result, it accepts the result and promised rewards are distributed to the workers.

challenge signed by client
If the client or one of the contracted workers thinks the evaluation result is wrong, it can request a re-evaluation to the parent node. A challenge requires an additional fee, but this fee is compensated by the evaluator's deposit in the event that the evaluator was truly wrong.

drop signed by admin

An admin node can drop one of its children if it thinks that the node is being dishonest. Dropped nodes cannot execute jobs any longer. Approving and dropping follows a certain rate limiting rule to prevent unexpected, injudicious drops.

send

A typical protocol to send coins between nodes.

4) Scenarios

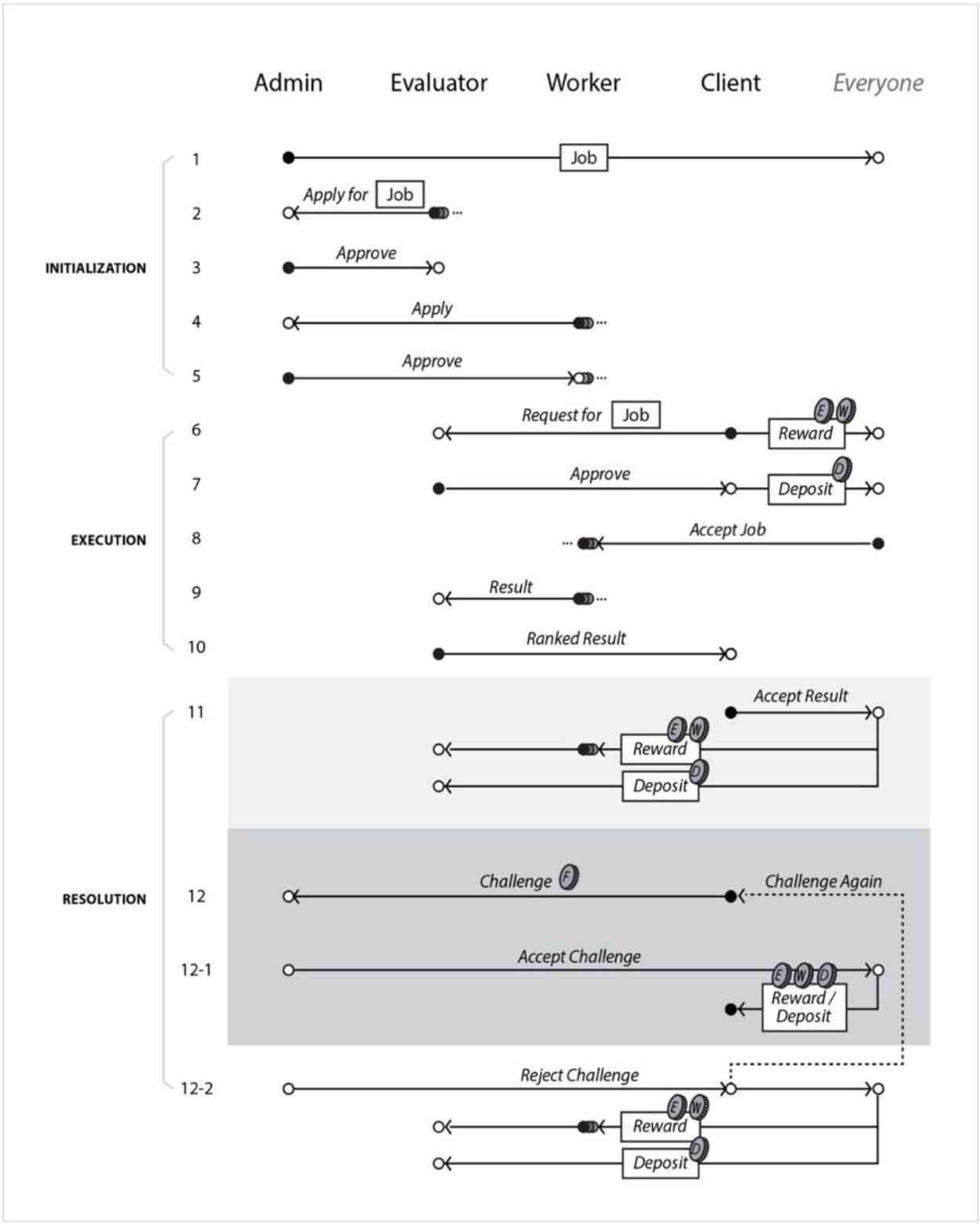

Figure 5. The diagram shows how jobs are registered and executed. The rewards for evaluators (E) and workers (W) are reserved in the network until the job is completed. Evaluator needs to deposit credit (D) to make sure it returns correct result.

Initialization Phase

1. Admin broadcasts a job schema to blockchain. The binary to be used for the job schema is published in Secure Runtime Environment, and related data are stored in Big Storage.
2. Evaluators who are interested in the job schema apply for the job schema. They download the binary from Secure Runtime Environment and data from Big Storage to test out whether they are capable of evaluating the job.
3. Admin approves Evaluators once it verifies Evaluators are eligible for the job schema.
4. Workers who are interested in the job schema apply for the job.
5. Admin approves eligible workers.

Execution Phase

6. Client requests a job with rewards for evaluators and workers. Rewards are reserved on the blockchain network until the solution is returned.
7. Evaluator approves the requested job and reserves deposit on the blockchain network.
8. Workers accept and start job execution.
9. Workers return the results.
10. Evaluators evaluate the results.

Resolution Phase

At this point, Client can either accept (11) or challenge (12) the solution.

11. Client accepts the result. The reserved rewards are distributed to workers and evaluators.
12. Client challenges the solution. If Admin accepts the challenge (12-1), Client receives reward and deposit. If Admin rejects challenge (12-2), Client can either challenge again or rewards are distributed to workers and evaluators.

5) Execution Details

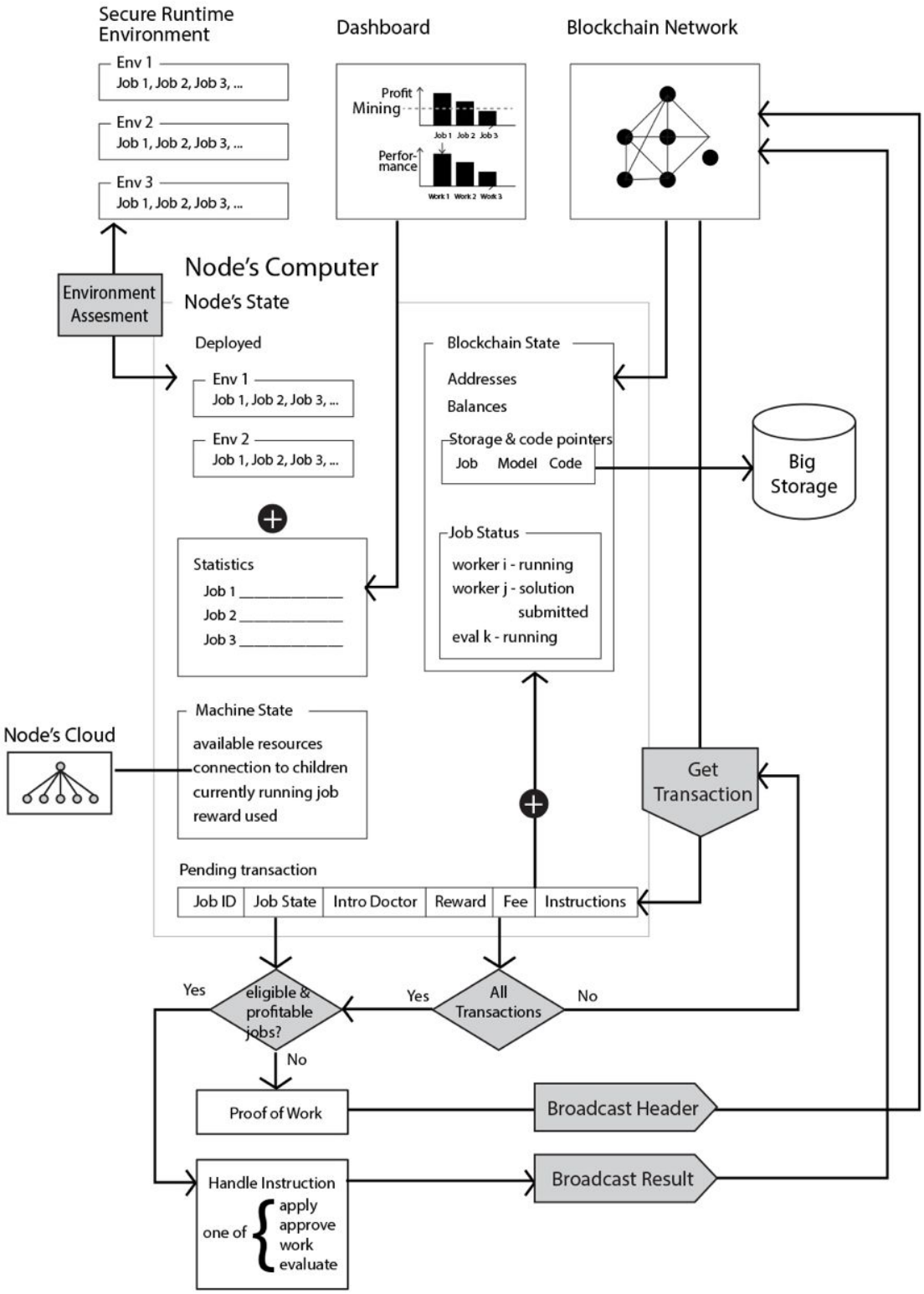

Figure 6. The diagram shows how jobs are evaluated or executed in a node's computer. The node identifies profitable jobs using blockchain and dashboard and participates in PoW if no profitable job is available.

When the node joins the blockchain network, it will start receiving all blocks from its peers to establish valid Blockchain State. After initialization, it will monitor transactions broadcasted through the network and update the Blockchain State. Blockchain State contains Job pointers and Job Status, and the node can use them to find suitable jobs it can run in Dashboard. If some jobs are promising, it downloads the environment and binaries from Secure Runtime Environment. Machine State tracks computational powers and resources available and filters jobs that can be executed immediately. The node receives updates about job statistics and decides which jobs to run. After the node finishes the job, it uploads the job result to the Big Storage and broadcasts a pointer to the blockchain network.

6) Configurable Permissions and Consensus Protocols

AI Network blockchain uses Proof-of-Stake (PoS) for its main consensus protocol, which is influenced by Tendermint's BFT-based consensus algorithm. A major difference between AIN blockchain and Tendermint in terms of consensus protocols is that on AIN blockchain, it is the rules that drive the consensus process. The rules also specify who has the permission to change the process. The rule-driven architecture and the tree structure of the state make having multiple consensus protocols on AIN possible. In other words, the main consensus protocol dictates how blocks are forged on the main chain; however, each shard of the blockchain that manages only a subtree of the global state tree has the freedom to configure its own consensus rules.

This means that if an app uses AIN blockchain as its database and wants to apply a different consensus algorithm for updating its db, for instance increase the number of validators, the nodes who have the permission to write the consensus rules for the shard may change the protocol within their network. The changes may need to be approved by a certain number of nodes that participate in maintaining the shard, depending on how the rules are set.

The configurability of consensus protocols respects the needs and decisions of the groups of nodes that want to manage particular segments of the global state. More importantly, configurable permissions and consensus protocols allow a new sharding mechanism to be adopted where each shard manages its own partition of the state tree and even the main blockchain doesn’t need to keep track of all the data in the partitions. Consequently, the time to reach consensus shortens, and the amount of data that a node has to store decreases significantly as well.

7) Dashboard

By analyzing the transaction data in blockchain, the AI Network team will provide statistics for nodes and jobs. Admins, evaluators, workers, and clients can refer to these extensive statistics to decide whether to apply for jobs or accept applications.

For example, metrics could be profitability of job, the number of jobs processed, response time, acceptance / rejection ratio, and the number of managed nodes. Information like performance ranking could also be made available. One of the rating systems we consider is Elo rating where relative computation power is measured when two machines worked on a job together. Also, metrics can be configurable, and each job owner may want to make their own metrics which meet the characteristics of their job.

2.2.4 Machine Learning Patterns on AIN protocol

AIN protocol's job definition is general enough to accept a large spectrum of inputs and outputs. Although it is not limited to ML purposes, we will discuss possible patterns of ML using AIN protocol to help understand its usefulness. The examples here are limited to basic principles, so it could be used with more sophisticated techniques in the practices.

1) Distributed Computing

Blockchain tracks the communication between nodes, but does not have features for job management or auto-scaling. Despite this, the ultimate goal of this infrastructure is to distribute and assign jobs efficiently. Broadly speaking, we can make use of this infrastructure for distributed computing in two ways. We will provide basic boilerplate code for distributing jobs.

First, a client can break down the job into small jobs and send multiple job requests to multiple targets. Tensorflow provides strong support for breaking down tensor graph into multiple pieces and runs the training on different devices or machines. Since sending job requests to blockchains is asynchronous and the result will be returned within a specified time limit, a client can contact with multiple evaluators to process the small jobs. If designed properly, even merging partial jobs into one can be done by using different job schemas on the blockchain. However, job distribution techniques should be implemented by a client using its own code. We believe our infrastructure is flexible enough to accommodate different types of job distribution methods. The AI Network team will provide references for how well-known distribution techniques can be implemented using Blockchain APIs.

Second, one node can be composed of multiple computers. One node doesn't have to be always one computer. To get popularity in the blockchain, some nodes want to maximize their power by using their own computation cloud. There might be different implementations for the same job schema if the ML environments are various. The AI Network team will also work hard to provide a centralized cloud for processing large ML problems. The blockchain is not the only mechanism for getting attention from clients. Each node can have its own marketing channel by exposing their node pointer externally from the blockchain. For example, the AI Network team can be trusted not only because of jobs processed in history, but also by other resources such as code repository, or well-structured homepages that reveal the developer's identity. However, AI Network team's node is also competing under the same rule, and there's no guarantee that it will become one of the more popular nodes on the blockchain.

2) Optimizing Loss function

Assume we want to optimize locally convex loss function L(w) where w is initial weight vector with random values. If workers use gradient descents, it will try L(w) -> 0 by randomly changing w with w := w – a (dL(w)/dw). Evaluator receives the results of workers and gives the highest reward to the worker who satisfied L(w) ~ 0.

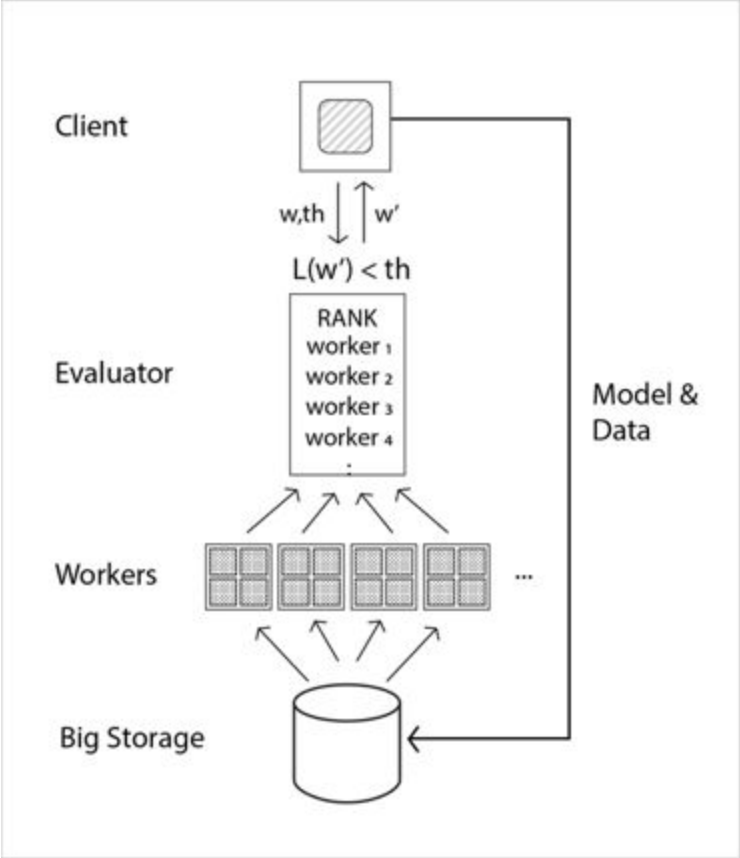

Figure 7. Optimizing loss function on AI Network

3) Data Parallelism

One way to achieve data parallelism using AIN protocol could be to use the client as a parameter server. Big Storage contains data shards of the input data that will be replicated in different locations such that data transfer time is optimized between workers and storage. In this scenario, the client asks multiple evaluators with information necessary for training such as pre-trained weights, model replica address, and data shard address, and evaluators will return the trained result. Then the client updates weights and sends the reconciled weights to the evaluators.

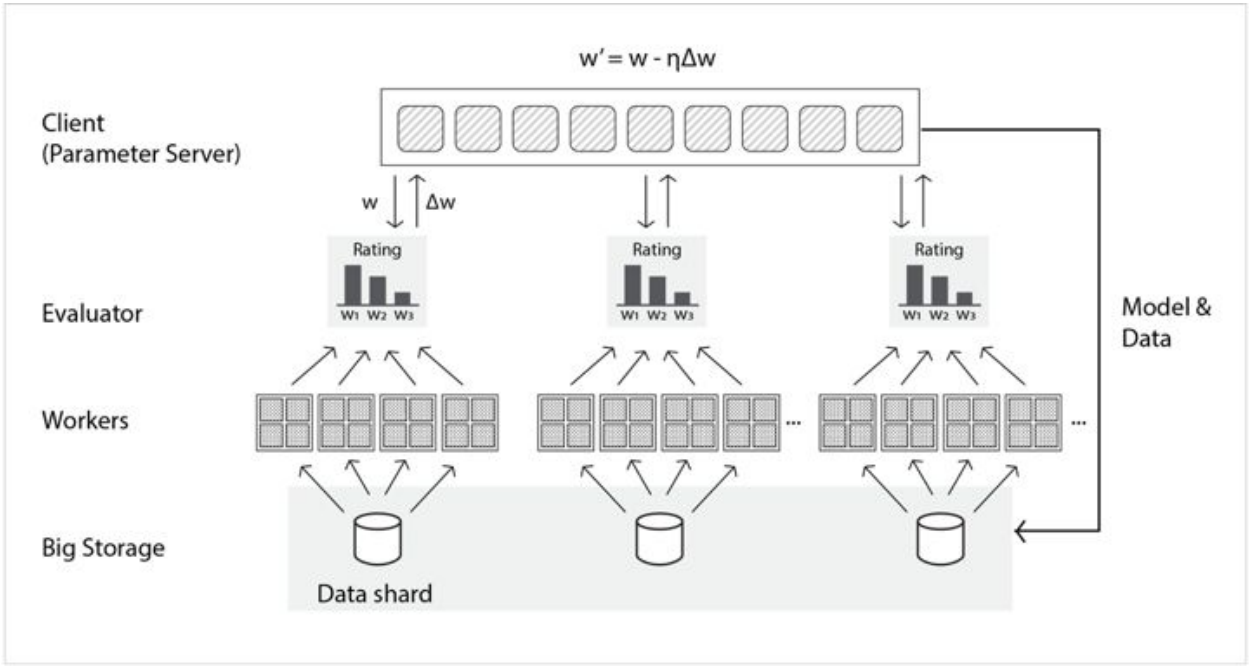

Figure 8. Parallel training on AI Network

4) Reinforcement training: AlphaGo case

In AlphaGo case, an evaluator can be a self-play function, where it conducts N rounds of self-plays with the policy network and the value network given by workers. The workers will be rewarded in the order of the ranking after self-plays. Then, the self-play data can be stored in Big Storage and shared with other workers for retraining purposes.

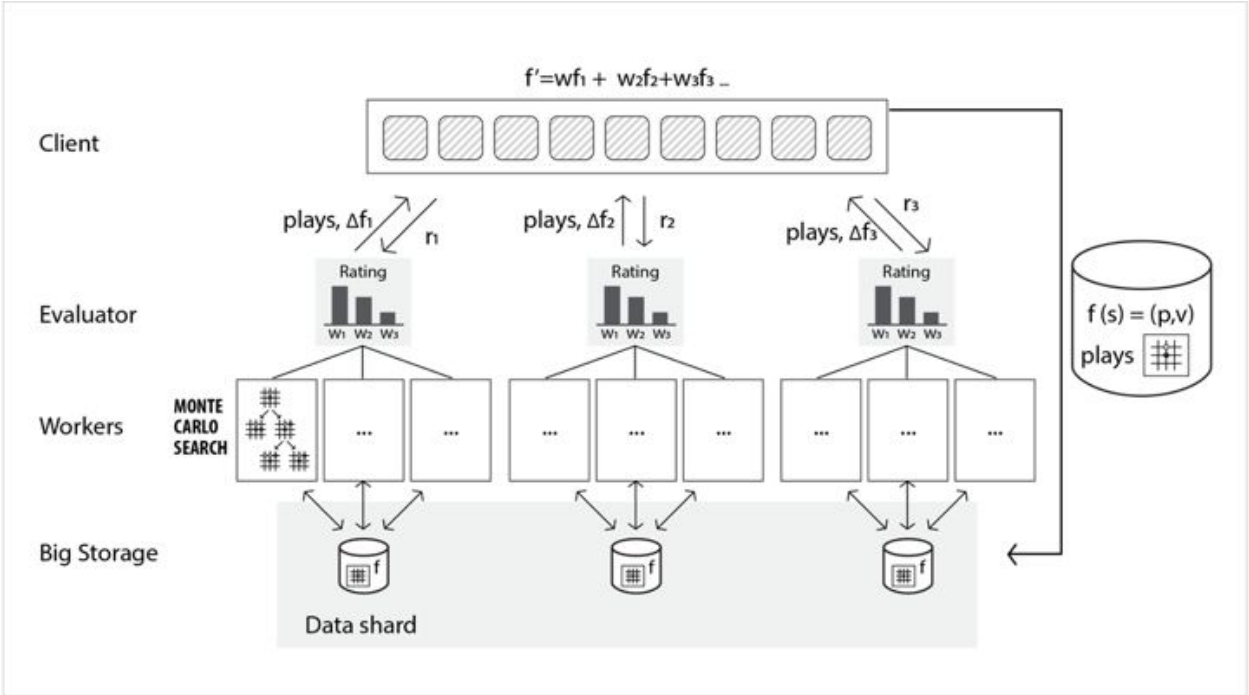

Figure 9. Reinforcement training on AI Network

5) Running custom code

Instead of deploying worker and evaluator codes, a client may want to get the execution result of custom code. In this case, evaluators' job is to make sure codes were run correctly by workers. Depending on the job's purpose and how evaluators and workers established their trust, the verification process may require full control over worker's hardware configurations, or verification functions or logs may need to be injected into the code. Establishing secure code execution environments is another big research topic, so we leave this up to the job schema's owner.

2.2.5 Security

1) Resource Protection

All nodes that run Blockchain are recommended to download docker container and binaries from AI Network team's repository. AI Network team will try hard to provide a trustful docker repository with various configurations. Currently supported ML configurations, libraries, and programming languages are well outlined in https://backend.ai/#/ground.

Docker restricts access and resource consumption over CPU, GPU, Network, Disk, and Memory. Although docker is a good distribution platform, it's not a security platform. It is recommended to run docker in isolated VM environment if user's PC contains sensitive information.

2) Binary Signing

A job schema must include a public code repository URL (GitHub, Bitbucket), and the repository should be built using AI Network team's builder with signing (i.e Google Container Builder). Through this signing process, we guarantee that the binary is built from our environment and that a code pointer is provided. However, it does not fully scan the code to detect maliciousness. It's up to the client to decide whether to trust publicly opened code or not. We recommend users not to run binaries from unknown sources unless the user has enough insight to distinguish whether the provided code is malicious or not.

3) Connection Security

All nodes in the blockchain communicate using the blockchain standard API. Docker Containers only allow blockchain's broadcasting channel and traffic from Big Storage if the job requires additional storage.

4) Data Security

Big Storage provides a global file name service to the blockchain nodes and is ACL-controlled by only valid nodes. We believe this will facilitate a service capable of competing in both performance and security with amazon s3 or google cloud storage.

However, since workers and evaluators both have access to the data, data preprocessing is always required in order to protect any sensitive information from these workers and evaluators.

For example, instead of providing raw data including sensitive information, shuffling and encrypting feature vectors would be safer. Also, we recommend input or neural networks size be modified to some meaningless number. For example, if the size of an input vector is 19 x 19, it would be very easy to guess you are solving Go problems.

We cannot prevent nodes from copying and transferring provided data. Therefore, if a client wants to evaluate workers with a separate data set from the training data, the test data set needs to be revealed at some future point in time. For example, if workers are provided with a training data set and evaluators are provided with a test set, the test set must be revealed after all the workers return their results. Otherwise, there is a possibility that an evaluator will expose the test data set in advance to a worker, and the worker can then try overfitting to the test data.

5) Compromised Node Scenarios

In this section, we explain attacking scenarios for each node type. Besides blockchain's security features, dashboard helps in detecting any fraud and punishing perpetrators of such fraud properly. However, there are certain security risks and loopholes that are out of scope in this architecture by design.

6) Admin

Admin nodes are the most powerful nodes in any job. Thus pose the greatest risk when compromised. Since the AI Network team manages binary releases of jobs in the docker repository, we recommend users to run only authenticated binary, especially in the early stages of the system.

An admin may attack the network by reproducing lots of admins beyond what it needs. However blockchain and P2P networks are known to be good at handling this type of denial of service attack. An attacker will need to pay large fees for making this large number of transactions.

If an admin is challenged a lot during job execution, people may think this job's processing is unstable and may contain malicious nodes. Using this approach, one may try lots of challenges to bring down an admin's reputation. However, making challenge requires fees and other nodes can always verify whether the challenge was legitimate since communication is open to everyone.

On the other hand, an admin may try to increase its reputation by solving self-generated valid jobs. However, it will require a large amount of fees to generate false operation, and the number of jobs solved is just one metric to gauge reputation.

7) Evaluator

If the evaluation result is not correct and the challenge from the client is successful, the client will receive the evaluator's deposit. The client cannot do DDoS attack on evaluators directly since there is no direct connection established between them. The evaluator and client are communicating using blockchain's broadcasting channel.

If an evaluator is not very responsive, the response rate of the evaluator in the dashboard will be bad, which results in a bad reputation. In addition, an evaluator runs the risk of being dropped by its parent if it is not working as expected.

There might be some cases where a good node is dropped without any reason. In that case, the dropped node can work alone instead of working for the parent node by being root of the same job. If the old admin is notorious for dropping nodes without any reason, people might prefer the newly created node.

8) Worker

Incorrect results from workers will not be accepted by evaluators, and it's also possible that the worker fails to give any results within the specified time limit. The worker who does not accept an evaluation result can initiate a challenge. However, if the workers challenge is not accepted after multiple attempts, the reputation of workers may also drop.

A client may assign very difficult jobs such that no worker can solve these jobs. This might be a useful strategy when the client only wants to know whether the size of the problem is solvable or not. In this case, the client will send a reward to the evaluator only. If this is not by job design, the one who designed the job needs to question its evaluation logic. If there are many nodes like this, workers will eventually lose interest in these jobs since they cannot earn proper and fair rewards. Good evaluation logic should be able to give proper rewards to workers who have honestly executed the provided job. Suppose there is an evaluator which accepts the answer if hash(x) = y, and there is a worker who runs x = rand(). If probability of hash(x) = y is very low compared to the reward, worker will not try to run this job anymore. In contrast, suppose f(x) = y and x = g(y), where f(x) = y is the evaluation function, and x = g(y) is the problem to be solved by the worker. If g is deterministic and can be easily run within polynomial time, it is attractive to workers since its reward is predictable.

3 Sample User Experience

Calculate Root

Given y, find x that satisfies x * x = y.
Suppose the worker is not smart enough so that it can only guess the answer using rand(). The client wants to give all the reward to the one who returned the closest answer. The simplest execution using console will look like the following:

```
$ connect AIN
AIN> registerJob job_schema.proto worker.py evaluator.py
Registration complete. Job ID is 0xabcde.
.. wait for propagation ..
AIN> stats 0xabcde
3 evaluators, 100 workers running.
AIN> requestJob {job_id: 0xabcde, job_input: {y: 4}, reward: [10, 10], time_limit: 3000}
X: 2
Total 12 workers participated. Evaluator (0xfghij) received 10, worker (0xklmn) received 10.
```

Sample job_schema.proto

```
// The greeting service definition.
service Worker {
  // Sends a greeting
  rpc CalculateRoot (Request) returns (WorkerReply) {}
}
service Evaluator {
  // Sends a greeting
  rpc EvaluateRoot (Request, Workers) returns (EvaluationResult) {}
}
message Request {
  float y = 1;
}

message WorkerReply {
  float x = 1;
}

message Worker {
  string node = 1;
  WorkerReply response = 3;
}

message Workers {
  repeated Worker workers = 1;
}

message EvaluationResult {
  repeated node = 1;  // returns highest ranked worker first. drops invalid worker.
}
```

Sample worker.py

```
class Worker(helloworld_pb2_grpc.WorkerServicer):
  def CalculateRoot(self, request, context):
          y = request.y
          min_loss = y
          for i in range(100):
          x = random.random() * request.y
          loss = abs(y - x * x)
          if min_loss > loss:
          min_loss = loss
        best_x = x
          return best_x
```

Sample evaluator.py

```
class Evaluator(helloworld_pb2_grpc.EvaluatorServicer):
  def EvaluateRoot(self, request, context):
          // sorting function omitted. Sorted by the closest answer.
          nodes = [worker.node for worker in sorted(workers)]:
          return nodes
```

```
// The greeting service definition.
service Worker {
  // Sends a greeting
  rpc CalculateRoot (Request) returns (WorkerReply) {}
}
service Evaluator {
  // Sends a greeting
  rpc EvaluateRoot (Request, Workers) returns (EvaluationResult) {}
}
message Request {
  float y = 1;
}

message WorkerReply {
  float x = 1;
}

message Worker {
  string node = 1;
  WorkerReply response = 3;
}

message Workers {
  repeated Worker workers = 1;
}

message EvaluationResult {
  repeated node = 1;  // returns highest ranked worker first. drops invalid worker.
}
```

Sample worker.py

```
class Worker(helloworld_pb2_grpc.WorkerServicer):
  def CalculateRoot(self, request, context):
        y = request.y
        min_loss = y
        for i in range(100):
        x = random.random() * request.y
        loss = abs(y - x * x)
        if min_loss > loss:
       min_loss = loss
        best_x = x
        return best_x
```

Sample evaluator.py

```
class Evaluator(helloworld_pb2_grpc.EvaluatorServicer):
  def EvaluateRoot(self, request, context):
        // sorting function omitted. Sorted by the closest answer.
        nodes = [worker.node for worker in sorted(workers)]:
        return nodes
```