CrowdCompute - A General-Purpose Peer-to-Peer Compute Engine with AI/ML extensibility

Mobin Hosseini
hosseini.mobin@gmail.com
Andreas Lymbouras
andreas.lymbouras@gmail.com
www.crowdcompute.io

July 11, 2020

Contents

Abstract

CrowdCompute is a peer-to-peer distributed compute engine network composed of unutilized computing resources like memory, general and graphics processing, and bandwidth. These pools of computing resources can be scaled both horizontally and vertically without any limitations and be used to run arbitrary code and workloads without any single point of failure. CrowdCompute's technical solution along with sharing economy principles forms an ecosystem that provides on-demand computing resources obtained in a peer-to-peer manner.

1 Introduction

Modern devices such as smartphones, tablets, personal computers, IoT devices, and servers are capable of running billions of instructions per second serving various programs. There are millions to billions of devices that are not utilized to their full capacity, with their processing capabilities sitting idle for long periods of time.

On the other end, there is an increasing demand for those resources. The increased demand is mainly driven by the recent trend of Artificial Intelligence and Machine Learning use cases in businesses that try to recognize patterns in their data.

CrowdCompute bridges the gap between unutilized computing resources and the demand for it. In this paper, we propose a solution to the use of unutilized resources using a peer-to-peer solution that marries providers' resources with the consumers who need them. The system can be used to run any sort of application, in a secure, and probabilistically verifiable way.

2 Background

This section covers the critical components and technologies needed for a scalable, fault-tolerant network for arbitrary code execution.

2.1 Public-key Cryptography and Hash Functions

Public-key cryptography uses a pair of keys called public/private key pair. They are considered a "pair" because the public key is derived from the private key. These keys are based on mathematical functions that have a special property: it is easy to calculate them, but hard to calculate their inverse. Meaning that even if you know the mathematical function that created the public key, you can't infer its private key.

This pair can be used to encrypt messages using the recipient's public key. The message can only be decrypted by the recipient holding the associated private key.

Public-key cryptography can also be used to verify the identity of a person sending a message using digital signatures. A sender using his private key in combination with the message to be sent can create a digital signature which can be verified by any receiver having the sender's mathematically associated public key. The digital signature here acts like a proof that the sender indeed holds the private key of associated public key.

In CrowdCompute we use public-key cryptography for both encrypting messages send amongst peers and verifying the identity of message senders.

A cryptographic hash function is a mathematical function that takes any size of bits (this is the input message — a text, an image, a number) and outputs a fixed length of 256-bits. This output is called the hash value, digest or simply the hash of the message.

Cryptographic hash functions have these properties:

1. It is infeasible to modify a message without changing the hash value. If you slightly change the input message, even one bit, the resulting hash value changes completely!

2. It is infeasible to find two different messages with the same hash value. The same input message (set of bits) will always give the same hash value (256-bits) and two different messages (sets of bits) will give different hash value (256-bits).

3. It is infeasible to generate a message from its hash value. Having the 256-bit hash value you can't go backward and find the message used as an input.

Hash functions such as KECCAK-256 are widely used within CrowdCompute both for verifying the integrity of data sent between peers, and uniquely identifying larger data like public keys, files, docker images, etc.

2.1.1 Elliptic-curve cryptography and ECDSA

Elliptic-curve cryptography (ECC) is a type of public-key cryptography based on the algebraic structure of elliptic curves over finite fields such as prime integers. We use a specific elliptic curve and set of constants, called secp256k1 which belongs to the category of Elliptic Curve Digital Signature Algorithms and defined by the following function:

$$y^2 = x^3 + 7$$

over a finite field of integers mod p where

$$p = 2^{256} - 2^{32} - 977$$

2.2 Distributed Hash Table

Distributed Hash Tables (DHT) are widely used in peer-to-peer systems to store the state and metadata of the peer-to-peer network. CrowdCompute's DHT implements the Kademlia specification with S/Kademlia modifications.

Kademlia uses a binary tree structure to represent each node in the network. Each node has a randomly generated 160-bit identifier which is hashed with SHA-1.

Kademlia defines the distance between two identifiers x and y as their bitwise exclusive or (XOR) interpreted as an integer, d(x,y) = x (XOR) y, thus:

$$d(x,x) = 0, d(x,y) > 0 \; if x \neq y, and \; \forall x, y : d(x,y) = d(y,x)$$

In other words, the distance of a node with itself is always zero unless the nodes are distinct, and the distance of two distinct nodes are always equal when measured individually by each node.

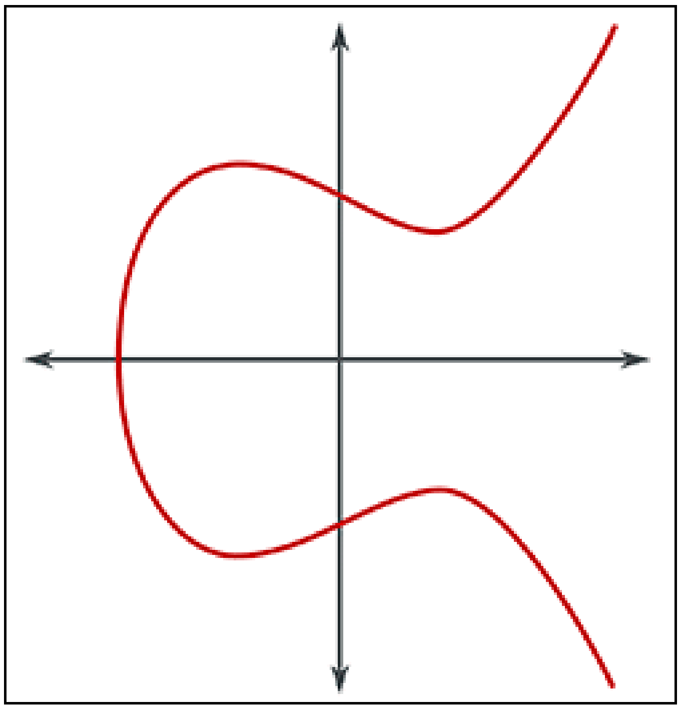

Figure 1: Elliptic curve

The algorithm needs to know the key identifier when looking for any value and explores the network in several stages. On each stage, it will find nodes that are closer to the key until the node has the value or no more closer nodes exist. This becomes more efficient compared to other DHTs as it contacts only O(log(n)) nodes when searching n nodes.

It also has further advantages associated with node failure tolerance and asynchronous, parallel querying making it a perfect candidate for a network consisting of millions of nodes.

2.3 Containers and Virtualization

In a p2p environment, the users' code should not be trusted at all, which leads to the need for proper implementation of a sandboxed environment.

Virtualization and containerization are types of sandboxed environments giving lots of benefits like:

- Language-agnostic

- Security and Isolation
- Performance

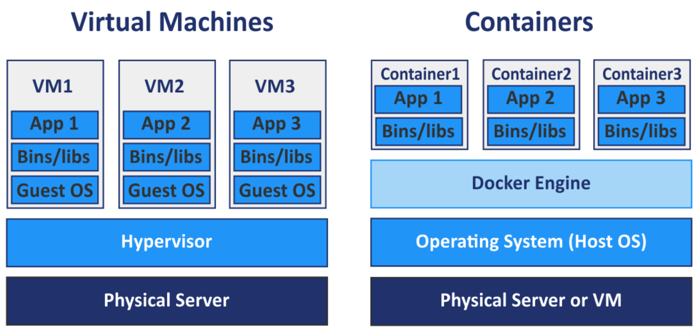

Figure 2: Virtualization vs containerization

2.3.1 Language-Agnostic

One of the main characteristics of containers is portability. Consumers can package their code and its dependencies into containers, without caring about the details of the operating system their code is being run on.

2.3.2 Security and Isolation

One of the challenges of Crowdcompute is to securely execute untrusted code without compromising the host's OS, filesystem, and networking stack. Applications running within containers are safer as they run in isolation from the machine they are hosted.

Running code within a container protects providers' machines as they run in isolation. You can read more here [9] about Docker containers' security.

2.3.3 Performance

Speed and performance are critical in a distributed computing system where various network delays can occur. The use of container engines significantly reduces the amount of time needed to provision and execute code. Lightweight virtualization or micro-VMs also provide similar speeds of deployment as well as offering better isolation.

2.3.4 Containers - Portable software

CrowdCompute abstracts the functionality of tasks (code execution), giving a variety of choices to users.

These being Docker containers, KVMs, AWS's Firecracker, or any other engine. This will attract users familiar with various Virtualization technologies but also give more options to the Consumer.

3 CrowdCompute Design

CrowdCompute nodes connect with each other to form a network. Nodes transfer messages to the network.

1. Identities
2. Network & Routing
3. Compute Engine
4. Blockchain & Consensus Mechanism
5. Reputation System
6. RPC

3.1 Identities

All the nodes in the network have public/private keypairs to validate the authenticity and integrity of the data they are communicating. This is a protective measure against attacks.

Public key encryption is used to ensure confidentiality since only private key owners can decrypt a message. And digital signatures are used to authenticate the senders' of a message as well as ensure the integrity of a message.

3.2 Network & Routing

There are three main actors within the CrowdCompute network. Providers, Consumers, and App developers.

1. Providers contribute the computational power, from laptops to data centers.

2. Consumers, individuals, or businesses, run tasks on the infrastructure offered by the providers.

3. App developers create open or closed source applications on the Apps marketplace. Consumers can use them for their tasks.

3.2.1 Peer-to-Peer Network Protocol

The Peer-to-Peer(p2p) protocol is a communications model run by all the peers in the network. In a decentralized network, each peer runs the same protocol (set of rules).

3.2.2 Network Formation & Node discovery

CrowdCompute is built on the Kademlia distributed hash table (DHT). Kademlia specifies the structure of the network as well as the exchange of information using the UDP communications protocol.

CrowdCompute expands Kademlia to:

- Signing packets
- 512-bit public keys as node IDs
- Removal of Hashtable-related features (finding and storing values within the DHT)

Newly joining peers should be able to discover and connect to the existing peers of the network.

Peer discovery can be achieved with the following methods:

1. Local node repository discovery
2. Predefined seed nodes
3. User-defined node addresses

4 Compute Engine

Computation is abstracted and decoupled from the rest of the system by introducing an additional layer that is responsible for the deployment and the execution of code and processes. The abstraction covers virtual machine deployment, containers, and root access.

5 Reputation System

Crowdcompute adopts a reputation system to minimize the adverse behavior of malicious or untrustworthy peers in the network.

Associating reputation with a particular node requires a sufficiently persistent identifier. We chose the node's wallet public key as an identifier because it provides several important properties like anonymity, spoof-resistance, and unforgeability.

Once a transaction on the network is completed, a reputation score is computed for the peers involved. Special agents called verifiers are responsible for giving a score to the parties involved in a transaction. Provider-nodes' honesty is being challenged by the verifiers as we will explain in more detail in a later chapter. The ability to solve those randomly generated challenges determines provider-nodes' overall reputation score.

5.1 Inputs - Outputs

Though it is important to consider both good and bad behavior, bad behavior should have a greater negative impact on the reputation score than the positive impact of a good one.

Each challenge's difficulty level could also be taken into account and have a weighted score.

Time could also be one of the parameters in the reputation score function. Recent transaction activity should outweigh older transactions when calculating the score. For example, a weighted transaction history could be used. This would allow validators to detect and defend against those peers who unexpectedly misbehave.

Ideally, for a provider-node, both its online availability and ability to prove challenges will be determined in two separate scores. In the end, a separation of scores will give a clearer view and distinction between highly available nodes and highly trusted ones.

5.2 Blockchain & Consensus Mechanism

To ensure the immutability of the reputation information and the state of the network, a blockchain is used with a proof-of-authority consensus algorithm.

5.3 RPC

JSON-RPC light-weight remote procedure call (RPC) protocol. It is transport agnostic in that the concepts can be used within the same process, over sockets, over HTTP, or in many various message passing environments. It uses JSON (RFC 4627) as a data format.

The JSON-RPC protocol is the way to talk to a CorwdCompute node.

6 Probabilistic Verifiable Computation

6.1 The problem

Sometimes, computing providers can be incentivized, for financial reasons, to skip the execution of the clients' code.

Verifying the execution of a software job is a known term called verifiable computing.

Verifiable computing gives consumers the ability to verify the correctness of the result of their job when being run on an untrusted provider.

6.2 The solution

We design a mechanism of verifying the execution of consumers' jobs to untrusted providers and provide its rigorous analysis.

Apart from clients and providers we also introduce a third entity, the verifiers. These are trusted entities whose job is to track providers' honesty and build trust in the network.

We achieve this by having a reputation system in place. Providers' reputation is calculated from their ability to solve challenges, while consumers' by their trustworthiness.

The lifecycle of a job is as follows:

1. New tasks are being created by consumers.
2. A client chose the workers he wants to cooperate with.
3. Verifiers compose challenges for the workers
4. Verifiers send those challenges to the client.
5. The consumers pass their tasks, along with the challenges, to the workers.
6. Worker nodes start executing the tasks and challenges.
7. Results are returned back to the client who passes them to the verifier.

8. The verifier checks the correctness of the challenges. If there is at least one unsuccessful challenge then the dishonest worker loses reputation and gains reputation otherwise.

We distinguish jobs in two types:

1. Running arbitrary software/code (Processes)
2. Full access to the machine (Remote connections, root shells, etc.)

Full access to a provider's machine will only be applied to fully trusted machines either configured by us or other trusted partners.

Incentives Failing to solve a challenge severely damages a provider's reputation which results in a lower probability of getting tasks from clients. So, a provider with a lower reputation gets fewer jobs, which results in less income.

Let's suppose attackers can create algorithms that can recognize challenges from actual jobs. Keeping challenges as identical as possible to the actual jobs poses difficulties and probably more wasted processing power running those algorithms than running the actual jobs. This is another economic incentive factor that drives providers to play honestly.

Challenges The challenges have to be identical to actual jobs that would be hard for anyone, machine or a human, to find differences.

This is achieved in a way where we continuously evaluate and improve our own challenges to keep up with smarter attackers.

They have to be balanced between:

1. Simple enough to run and verify
2. Hard enough to solve challenges without executing

Calculations Computation of providers' reputation is being calculated by this algorithm:

$$g(x) = \begin{Bmatrix} 1 & if & fw(cx) = fv(cx) \\ 0 & if & fw(cx) \neq fv(cx) \end{Bmatrix}$$

$$f(x) = \begin{Bmatrix} 0 & if & (\sum_{i=1}^{N} g(i) < 1)/N \\ 1 & & otherwise \end{Bmatrix}$$

$$rep = (\sum_{j=1}^{M} f(j))/M$$

7 Attacks

7.1 Denial of Service

An attacker could initiate Denial of Service(DoS) attacks to prevent the system from responding to authentic requests, and keep the system resources busy without doing anything useful. Nodes under DoS attacks are treated as offline nodes and don't impose any issues as long as there are a few online nodes. It's a good practice to isolate nodes in a subnode which is monitored and protected by IDS and firewalls.

7.2 Kademlia Spartacus

An attacker joins the network claiming to have the same identity as another node is classified as a Spartacus attack. An attacker node could simply copy the node id and forward a message to another node commanding them to perform certain actions. This is addressed by implementing node ids as ECDSA public key hashes and requiring messages to be signed.

7.3 Sybil Attack

Sybil attacks are performed when an attacker creates a huge set of nodes to disrupt the operation of the network simply by discarding the signed message sent to the neighbors. Neighbor nodes are selected in a randomized, evenly distributed pool, and messages are sent to at least three neighbors.

8 Conclusion

We have proposed a system for distributed execution of arbitrary code. We started with the presentation of existing technologies that are used in CrowdCompute and continued to talk about CrowdCompute's design. Digital signatures and public-key cryptography, provide strong control of ownership and authenticity of digital messages in the peer-to-peer network communication. DHT is used for storing various metadata for the network, while containers and virtualization for the language-agnostic code execution along with their performance and strong security and isolation nature.

However, it is incomplete without a way to verify code execution. To solve this, we proposed an algorithm where Verifiers estimate the worker's credibility by giving them challenges and attaching a reputation based on their ability to solve them.

References

[1] Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019,
`https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-wo`

[2] B. Cohen. Incentives build robustness in BitTorrent, (2003).
`http://www.bittorrent.org/bittorrentecon.pdf`

[3] P. Maymounkov, D. Mazieres. Kademlia: A peer-to-peer information system. based on the xor metric, (2002).
`https://pdos.csail.mit.edu/ petar/papers/maymounkov-kademlia-lncs.pdf`

[4] J. Poon, T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, (2016).
`https://lightning.network/lightning-network-paper.pdf`

[5] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3), (2017).
`https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pd`

[6] Peng Wang, James Tyra, Eric Chan-Tin, Tyson Malchow, Denis Foo Kune, Nicholas Hopper, Yongdae Kim. Attacking the Kad Network, (2009).
`https://www-users.cs.umn.edu/h̃oppernj/kad_attack_securecomm.pdf`

[7] Sergio Marti. Trust and reputation in peer-to-peer networks, (2005).
`https://crypto.stanford.edu/portia/papers/marti.pdf`

[8] Satoshi Nakamoto. A Peer-to-Peer Electronic Cash System, (2008).
`https://bitcoin.org/bitcoin.pdf`

[9] Docker security.
`https://docs.docker.com/engine/security/security/`