Santiago de Cali, 4 de Julio del 2018

Doctor:
Juan Carlos Martinez
Director Posgrados de Ingeniería
Facultad de Ingeniería
Pontificia Universidad Javeriana Cali

Habiendo cumplido los requisitos establecidos en los artículos 5.6 y 5.7 de las Directrices para Trabajo de Grado de Maestría, solicitamos se autorice la sustentación del Trabajo de Grado denominado "Computación Evolutiva Descentralizada de Modelo Híbrido usando Blockchain y Prueba de Trabajo de Optimización" realizado por el (la) estudiante Harvey Demian Bastidas Caicedo con código 0022445 perteneciente al énfasis en Ingeniería de Sistemas y Computación, bajo la dirección de la profesora María Constanza Pabón Burbano.

El suscrito director del Trabajo de Grado autoriza para que se proceda a hacer su sustentación ante el Tribunal que para el efecto se designe, toda vez que ha revisado meticulosamente el documento y avala que el Trabajo de Grado ya se encuentra listo para ser evaluado oficialmente.

Atentamente,

Harvey Demian Bastidas Caicedo	María Constanza Pabón Burbano
C.C. 94526156 de Cali	C.C. 34.559.226 de Popayán

Documentación anexa:

Tres copias anilladas del documento de Trabajo de Grado, con impresión por lado y lado y paginación completa.
El resumen del Trabajo de Grado en formato electrónico (máximo 1 página).

DATOS DEL ESTUDIANTE

Nombre Completo: Harvey Demian Bastidas Caicedo
Dirección: Cra 56 # 3 -88 Apto 601 – Cali, Colombia
Teléfono: (+572) 378 30 80
Celular: (+57)3006200759
Email: harveybc@ingeni-us.com
Profesión: Ingeniero Electrónico
Universidad: Universidad de San Buenaventura - Cali
Empresa: Ingeni-us Soluciones de Ingeniería S.A.S.
Cargo: Fundador - Director Ejecutivo

FICHA RESUMEN

TRABAJO DE GRADO DE MAESTRÍA

TITULO: “Computación Evolutiva Descentralizada de Modelo Híbrido usando Blockchain y Prueba de Trabajo de Optimización”

1. **ÉNFASIS**: Ingeniería de Sistemas y Computación
2. **ÁREA DE INVESTIGACIÓN**: Inteligencia Artificial, Computación Evolutiva, Optimización
3. **ESTUDIANTE**: Harvey Demian Bastidas Caicedo
4. **CORREO ELECTRÓNICO**: harveybc@ingeni-us.com
5. **DIRECTOR**: María Constanza Pabón Burbano
6. **CO-DIRECTOR(ES**): Ninguno
7. **GRUPO QUE LO AVALA**: Ninguno
8. **OTROS GRUPOS**: Ninguno
9. **PALABRAS CLAVE**: Optimization, Evolutionary Computation, Neuroevolution, Genetic Algorithm, Peer-to-Peer, Blockchain, Proof of Work, Reinforcement Learning, Decentralized Networks, Distributed Computing , Neuroevolution, Bitcoin
10. **CÓDIGOS UNESCO CIENCIA Y TECNOLOGÍA**: 1203.04, 1203.18, 5310.04
11. **FECHA DE INICIO**: 1 de enero de 2017 DURACIÓN ESTIMADA: 18 Meses
12. **RESUMEN**: En este documento se propone el uso de un blockchain con una prueba de trabajo de optimización (OPoW) para implementar un servicio de timestamping en una red descentralizada de optimización con Computación Evolutiva de modelos híbridos, usando para optimización la capacidad computacional que es usada para la generación de una prueba de trabajo criptográfica en otras redes basadas en blockchain como Bitcoin. El sacrificio de usar una prueba de trabajo útil es que la OPoW no es una función del contenido del bloque, sino solo del estado de optimización. Para la validación empírica de la OPoW, este documento describe el diseño de una plataforma de software descentralizada para implementar Algoritmos Evolutivos Distribuidos (dEA) utilizando el modelo de isla y los modelos híbridos.

RESUMEN

Las técnicas de *Computación Evolutiva (EC)* como *algoritmos genéticos, neuroevolución o swarm intelligence* son métodos de optimización de parámetros de modelos matemáticos que se caracterizan por el uso de una población de soluciones candidatas que evolucionan en un espacio de búsqueda de una forma inspirada en los principios de la evolución biológica como la competencia, la selección o la reproducción. Existen varios modelos arquitecturales para implementar técnicas de EC en arquitecturas de procesamiento distribuido (*dEC)*, los modelos con mejor tolerancia a fallas y menor costo comunicacional son los *modelos híbridos* basados en el *modelo de Islas.*

Múltiples modelos para *dEC* se han implementado en frameworks o plataformas de software, pero las implementaciones encontradas tienen desventajas como que tienen baja tolerancia a fallas o les faltan mecanismos de trazabilidad que podrían ser deseables o necesarios para algunas aplicaciones. Para contrarrestar las desventajas mencionadas, el *blockchain* y la *prueba de trabajo criptográfica (CPoW)* son tecnologías para almacenar datos de trazabilidad de eventos en redes descentralizadas, pero con el requerimiento de una capacidad computacional adicional para la generación de una *CPoW.*

En este documento se propone el uso de un blockchain con una prueba de trabajo de optimización (*OPoW*) para implementar un servicio de *timestamping* en una red descentralizada para optimización con *dEC* de modelos híbridos, usando para optimización la capacidad computacional que es usada para la generación de una prueba de trabajo criptográfica en otras redes basadas en blockchain como Bitcoin. El sacrificio de usar una prueba de trabajo útil es que la OPoW no es una función del contenido del bloque, sino solo del estado de optimización.

Para la validación empírica de la OPoW, este documento describe el diseño de una plataforma de software descentralizada para implementar *Algoritmos Evolutivos Distribuidos* (*dEA*) utilizando el modelo de isla y los modelos híbridos. La plataforma propuesta explora el uso de un *blockchain* y una prueba de trabajo de optimización para almacenar un registro de operaciones para la trazabilidad y la sincronización de los estados de optimización de los nodos participantes en los procesos de *dEC*.

La plataforma propuesta fue implementada y una aplicación para aprendizaje por refuerzo usando dEC en el dominio de automatización de comercio de divisas se usó para realizar experimentos para validar la escalabilidad, tolerancia a fallos y rechazo de resultados inválidos que provee el uso de OPoW.

Palabras clave*:* *Optimization, Evolutionary Computation, Neuroevolution, Genetic Algorithm, Peer-to-Peer, Blockchain, Proof of Work, Reinforcement Learning, Decentralized Networks, Distributed Computing, Neuroevolution*

ABSTRACT

The *Evolutionary Computation* (EC) techniques such as genetic algorithms, neuroevolution or swarm intelligence are optimization methods characterized by using a population of candidate solutions that evolve in a search space in a way inspired by biological evolution principles like competition, selection or reproduction. There are several architectural models for implementing EC techniques in distributed processing architectures (*dEC*), the models with better fault tolerance and lower communicational cost are the *hybrid models* based on the so-called *island model*.

Multiple models for *dEC* have been implemented in frameworks or software platforms, but the existing implementations are either programming language-specific, lack fault-tolerance or lack traceability features that could be desirable or required for some applications. For counteracting the mentioned disadvantages, the *blockchain* and the hash-based *cryptographic proof-of-work (CPoW)* are technologies that allow the storage of data for the traceability of events in a decentralized network, but with an additional computational capacity requirement for the generation of a CPoW.

This document proposes the use of a blockchain with an *optimization proof-of-work (OPoW)* to implement a timestamping service in a decentralized network for *dEC* with hybrid models, using for optimization the computational capacity that is used for hash-based proof-of-work generation in other blockchain based networks like Bitcoin. The tradeoff of a useful proof of work is that the *OPoW* is not a function of the block contents but only of the optimization state.

For the empirical validation of the proposed proof of work, this document describes the design of a decentralized software platform for implementing *distributed Evolutionary Algorithms (dEA)* using the island model and hybrid models. The proposed platform explores the use of a blockchain and an optimization proof-of-work to store a log of operations for traceability and synchronization of optimization states of the participating nodes in *dEC* processes.

The proposed platform was implemented and an application for reinforcement learning using dEC in the domain of foreign exchange trading automation was used to perform experiments to validate the scalability, fault-tolerance and invalid result rejection capabilities provided using OPoW.

Keywords*: Optimization, Evolutionary Computation, Neuroevolution, Genetic Algorithm, Peer-to-Peer, Blockchain, Proof of Work, Reinforcement Learning, Decentralized Networks, Distributed Computing, Neuroevolution*

DESCENTRALIZED HYBRID-MODEL EVOLUTIONARY COMPUTATION USING BLOCKCHAIN AND OPTIMIZATION PROOF OF WORK

HARVEY DEMIAN BASTIDAS CAICEDO

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
SANTIAGO DE CALI
2018

DESCENTRALIZED HYBRID-MODEL EVOLUTIONARY COMPUTATION USING BLOCKCHAIN AND OPTIMIZATION PROOF OF WORK

HARVEY DEMIAN BASTIDAS CAICEDO

Proyecto de trabajo de grado de Maestría en Ingeniería

Director
María Constanza Pabón Burbano

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
SANTIAGO DE CALI
2018

TABLE OF CONTENTS

1. Introduction 1
2. Problem Definition 3
   2.1. Problem Statement 3
   2.2. Scope and Limitations 4
3. Objectives 5
   3.1. General objective 5
   3.2. Specific Objectives 5
4. Justification 6
5. Theoretical Framework and State of the Art 8
   5.1. Platform Topics 8
      5.1.1. Evolutionary Computation (EC) 8
      5.1.2. Distributed Evolutionary Computation (dEC) 9
      5.1.3. Decentralized Architectural Pattern 12
      5.1.4. REST Application Programming Interface (API) 13
      5.1.5. Blockchain 14
      5.1.6. Cryptographic Proof of Work 15
      5.1.7. Useful Proof of Work 16
   5.2. Prototype and Experiment Topics 17
      5.2.1. Reinforcement Learning (RL) 17
      5.2.2. Automated Foreign Exchange Trading 19
6. Optimization-Based Proof-Of-Work 22
   6.1. Proof-Of-Work Security 22
   6.2. Useful Proof-Of-Work 22
   6.3. Optimization Proof-Of-Work 23
   6.4. Block-Time Control 23
   6.5. Empirical Validation 24
7. Platform Design 26
   7.1. Requirements 26
      7.1.1. Stakeholders 26
      7.1.2. User Story: Forex Trading Automation 27
      7.1.3. Non-Functional Requirements 29
      7.1.4. Functional Requirements 29
   7.2. Components and Behaviours 30
      7.2.1. Usage of Nodes 32

7.2.2. Usage of Optimizers 33
7.2.3. Usage of Evaluators 33
7.2.4. Usage of Clients 33
7.3. Design of Identified Components 33
7.3.1. Node 33
7.3.2. Optimizer 39
7.3.3. Evaluator 41
7.3.4. Client 43
8. Technical Decisions 45
8.1. Development and experimental Environment 45
8.2. Source Code Version Control 45
8.3. Programming Languages 45
8.3.1. Nodes 45
8.3.2. Optimizers, Evaluators and Clients 46
9. Validation Experiments 47
9.1. Experiments Description 47
9.1.1. Scalability Experiment 49
9.1.2. Fault-Tolerance Experiment 50
9.1.3. Invalid Result Rejection Experiment 51
9.2. Experimental Setup 51
10. Results and Analysis 53
10.1. Scalability Experiment 53
10.2. Fault-Tolerance Experiment 53
10.3. Invalid Result Rejection Experiment 53
11. Discussion 55
12. Future work 57
13. Conclusions 58
14. References 59

LIST OF FIGURES AND TABLES

Figure 1 - Evolutionary Algorithms Structure 9
Figure 2 - Distributed EC Framework. 100
Figure 3 - Population-distributed and dimension-distributed EC models 111
Figure 4 – The Agent-Environment Interaction in Reinforcement Learning 177
Figure 5 – Difference between centralized and decentralized architectural patterns..... 300
Figure 6 - Expected behavior of the components of the proposed platform. 311
Figure 7 - Platform Architecture 355
Figure 8- Component-level Architecture of Nodes 39
Figure 9 - Network configuration for the fault-tolerance experiment 50
Table 1 - Scalability Experiment Results 53
Table 2 - Fault-Tolerance Experiment Results..... 53
Table 3 – Invalid Result Rejection Experiment Resuts..... 54

1. INTRODUCTION

Evolutionary Computation(EC) techniques are used to search for optimum values of mathematical model parameters [1] [2] [3]. As the search for solutions in EC is based on trial and error, optimization of complex models may require the use of a large computational capacity for testing many candidate solutions or for processing large datasets [4] [5].

The defining characteristic of an *Evolutionary Algorithm (EA)* is that an optimization process state is composed of a population of candidate solutions called *specimens* representing points in a search space. These specimens are evaluated to determine their performance in some task and then they are slightly but randomly modified to be tested in a different location in the search space during the next iteration.

The implementation of an EA in a distributed architecture, can be made using many existing architectural models, but the one that provides the best fault-tolerance and lower communicational cost is *the island-model* [6]. The island model is decentralized and executes parallel processes with independent populations in each network node and they only communicate to each other with a *migration* operator that merges the best performing specimens of each process to propagate the best solutions through the network. The island-model is also used as higher layer in model ensembles in the so-called *hybrid-models.* Existing frameworks implementing *hybrid-models* exists, but the ones known by the author are either programming language-specific or lack fault-tolerance or traceability features that can be desirable or required for some optimization applications over insecure environments.

A large motivation for this work is the fact that EC techniques can be applied to a wide spectrum of real world applications, including: systems design [7], simulation optimization [8], resource scheduling [9], network planning [10], feature extraction [11] and parameter tuning of machine learning models such as neural networks [12]. Some of these applications may benefit from having a decentralized architecture as an alternative option to the existing centralized architectures, since it provides scalability and fault-tolerance. However, existing implementations on decentralized architectures do not have a way to perform traceability of the operations made by participating devices limiting the scope of the possible applications.

In decentralized networks, a node can verify the existence of some information stored by all the other nodes but since nodes can have different time settings, it can't verify the date of creation of the information and it neither can't verify if the information has been changed since it was first created on a node. As an improvement over the existing island-model implementations, this work proposes a platform that uses blockchain for adding the possibility of use traceability and read-only storage in decentralized applications.

The blockchain and the Cryptographic Proof of Work (CPoW) are technologies used in Bitcoin that allow data storage and traceability of events in decentralized networks, but with the disadvantage that the generation of a CPoW requires the use of extra computational capacity.

This document proposes the use of a blockchain with an optimization proof of work (OPoW) instead of a CPoW to implement dEC with hybrid models, using for optimization the computational capacity that otherwise would be used for the generation of a CPoW. It will be validated that the use of blockchain with OPoW allows storage and traceability in applications that use decentralized dEC, with scalability and fault tolerance without the additional computational cost of generating a CPoW.

The proposed blockchain stores data about synchronization and traceability of the optimization states of participating nodes, but additionally it can store any other data. The data stored in the blockchain can be different from the produced during the optimization process, it is read-only, and it has a timestamp independent from the time settings configured on the machine that generates the information, since it uses a decentralized timestamping service like the used in Bitcoin.

To carry out the validation of the blockchain with *OPoW*, a prototype of a software platform with an application for reinforcement learning was designed and implemented using *dEC* in the domain of foreign currency exchange automation using *OPoW*. The validations are limited to a single working application in the forex trading automation domain, that stores optimization and remote evaluation information but also authentication, authorization and accounting data in the blockchain. Different optimization processes can be used to produce the *OPoW* and non-related information such as security logs can be saved in the blockchain alongside with the optimization information used to create the blocks.

2. PROBLEM DEFINITION

This section contains the problem statement, and the scope and limitations of a proposed solution.

2.1. PROBLEM STATEMENT

In a decentralized network, nodes can have different time settings and because of that, a node can't verify the date of creation of some information that was made on other nodes and it neither can't verify if some information has been changed since it was first created on a different node. This problem limits the creation of decentralized applications with traceability, to overcome this limitation, a blockchain as the one used in Bitcoin can be used since it timestamps blocks of data using a CPoW to control the block time (period between blocks).

The CPoW is easy to validate but requires a high computational capacity to be generated. This computational capacity is spent in a cryptographic task that is designed to take approximately the desired block time, varying the difficulty of the task depending on the quantity of participating nodes. The process of producing a CPoW is called mining and it consumes electric energy that is ultimately turned into CO_2 and constitutes the carbon footprint of the CPoW generation in any application that uses it.

For some applications, a way to attenuate this problem is to employ a useful task in the generation of the proof of work instead of the cryptographic task, so the same carbon footprint is generated, but the task provides some useful calculus to compensate for it. An ideal case would be that the proof of work is produced by the same task that is producing the information to be saved in the blockchain, since the carbon footprint increment due to the use a proof of work is reduced to zero.

The author hypothesizes that an OPoW can be used as alternative to CPoW in applications that use optimization with EC in decentralized architectures, using for optimization the computational capacity that would otherwise be used to generate a CPoW. Additionally, the author proposes a platform that allows the use of optimized model parameters to evaluate data while the optimization is being executed.

The problem is to validate that it is possible to use a blockchain with OPoW instead of a CPoW to implement dEC with hybrid models and while both use computational capacity to control the block time, the OPoW generates useful results (optimized parameters of a mathematical model).

It is required also to validate if for the same optimization process to reach a known performance, the use of CPoW has a larger carbon footprint than the use of OPoW.

2.2. SCOPE AND LIMITATIONS

This project proposes a platform that uses OPoW, includes the design and implementation of the proposed platform components and the empirical validation of the scalability and tolerance to failures of a distributed optimization process. The validation is limited to a single working application in the forex trading automation domain.

3. OBJECTIVES

3.1. GENERAL OBJECTIVE

To design, implement and test a *prototype* of a software platform for decentralized optimization using dEC, blockchain for traceability, and an optimization proof-of-work to implement a decentralized timestamping service; and to deploy an application for optimization to validate this platform.

3.2. SPECIFIC OBJECTIVES

- To design a platform for decentralized dEC with traceability, the design includes describing its requirements, components, relationships, desired behavior, and data structures.
- To define communications protocol between participating nodes with the objective of achieving the desired behavior of the platform. The information exchanged in this protocol will be used for synchronization of nodes participating in an EC process.
- To implement a prototype platform with nodes that listen to REST API requests from other nodes to form a decentralized network.
- To implement an application for distributed EC that uses the proposed platform in the foreign exchange trading automation domain.
- To perform tests to validate the scalability, fault-tolerance and traceability in an application using the proposed software platform.

4. JUSTIFICATION

Distributed EC has many fields of real-world application, some applications include but are not limited to: systems design, for example, VLSI routing [7], simulation optimization [8], resource scheduling such as cancer treatment scheduling [9], network planning [10], traffic estimation [13], classifier optimization [14], feature extraction [11] and parameter tuning of machine learning models such as neural networks [12]. This variety of applications justify the effort to improve existing implementations to overcome their downfalls and to provide the *reliability* that a platform for industrial applications over insecure networks requires.

Fault-tolerance can be desirable in applications in which interruption can be costly, for example during forex trading automation or during a large optimization process. The *fault-tolerance* of an application can be implemented by using a decentralized architectural pattern, since decentralized networks are tolerant to faults in any of their nodes and provide *scalability*.

Having *traceability* of events (requests and responses) during dEC in the proposed platform allows the developers to debug a decentralized dEC application because the developers have an historical account of how the participating devices contributed to a decentralized optimization process or when they failed.

Another reason to implement *traceability* in a decentralized optimization application is that the historic log of operations performed by the devices in the network can be used to bill and pay for the services consumed and provided by participating devices. Also, is desirable for its implementation a language agnostic programming interface, so the working nodes can use different implementations of an EA in heterogeneous hardware.

The possibility of using the blockchain as a *traceability* mechanism for decentralized optimization with EA is the primary justification for this work. As far as we know, this proposal is the first use of the blockchain technology for distributed EA. A blockchain provides the consensus, traceability and trust-less node collaboration that can be used in distributed EA to make the results *reliable* in processes executing over public or insecure networks, as the Internet.

The possibility to employ a useful task in the proof of work generation, instead of the cryptographic task, is another motivation, since the same carbon footprint is generated for proof of work, but the task provides some useful calculus to compensate for it. Thus, the carbon footprint due to the use a proof of work is reduced to almost zero.

Another justification is the fact that while some distributed EA frameworks for particular languages as Python [15] or Matlab [16] exist, the ones known by the author are either

not language-agnostic, fault-tolerant nor have the security and traceability features that a production application would require.

The proposed platform allows to use a decentralized architecture with traceability for optimization processes that use dEC, providing fault-tolerance and scalability to an optimization process, a combination of features that may be useful for some applications but that existing solutions known to the author do not have.

A foreign exchange trading automation experiment will be performed as an application of distributed EC due to the author's previous experience with this problem domain.

5. THEORETICAL FRAMEWORK AND STATE OF THE ART

In the following subsections, the theoretical framework and the state of the art related to the proposed platform and prototype design topics are described, and the existing distributed EC platforms are discussed. The described platform topics are: evolutionary computation, decentralized architectural pattern, public-key cryptography and blockchain. The prototype experiment topics are: reinforcement learning and automated foreign exchange.

5.1. PLATFORM TOPICS

In this section the topics related to the design and development of the platform are described. *Evolutionary* computation is the set of techniques that use evolutionary algorithms to perform optimization of parameters of some deterministic mathematical model, the REST Application Programming Interface allows language-agnostic implementation of evolutionary algorithms in heterogeneous hardware, the *decentralized architectural pattern* uses REST and allows fault-tolerance in any node of the optimization network, and *blockchain* that allows traceability in a decentralized network.

5.1.1. Evolutionary Computation (EC)

The proposed platform was designed specifically for distributed optimization using Evolutionary Computation techniques. EC is a field of artificial intelligence that uses global optimization algorithms called *Evolutionary* Algorithms (EA) that are characterized by imitating Darwinian principles such as reproduction, mutation, recombination, natural selection and survival of the fittest [1][2][3].

The first EA dates from 1975, when Holland defined the first *Genetic Algorithms* (GA) whose objective was to search through a parameter space for a set of parameters that optimize some fitness criterion. In a GA, a population of points in the search space is created representing candidate solutions with a measurable distance between them. A string of parameters is called a *genotype* or a *specimen* and it is feed to a gene-expression function to produce the *phenotype* which is evaluated on the fitness criterion for a task.

The phenotypes with higher fitness are left unchanged between iterations and they are allowed to mate more times to create offspring with better parameterization. Mutations occur in every iteration and cause small random perturbations of parameters and ensure that the search covers new areas of the search space.

There are other evolutionary algorithms such as swarm particle optimization [17] that do not use genetic operators, the population in them is a set of particles with a position representing the parameter values to be found and speed changes represent the variations that allow the search of parameter values to continue until some fitness function is satisfied.

Other algorithms like Artificial Bee Colony [18] and Ant Colony Optimization [19] use nature-emulating mechanisms to control the movement of the specimens in the population.

A diagram of the structure of any evolutionary algorithm is shown in Figure 1. All the algorithms share a population initialization phase and have a cycle composed of performance *evaluation*, *selection* of the fittest and *variation* of the population so it covers new areas of the search space.

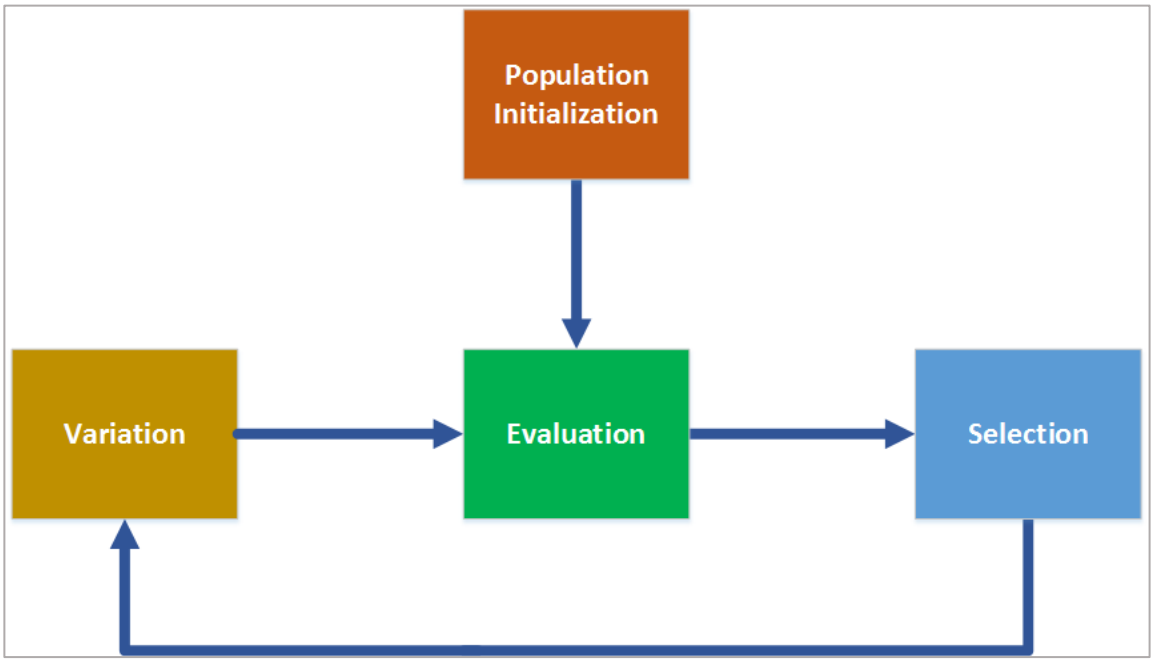

Figure 1 - Evolutionary Algorithms Structure

Neuroevolution Algorithms are a particular set of GA for optimizing neural network parameters. A Neuroevolution algorithm will be used in the proposed application experiments.

Recently, the global search capabilities of EA are used to search parameters of Machine Learning techniques for example: Support Vector Machine parameter estimation with Genetic Algorithms [20], Deep Learning [21] [22] and Neuroevolution for estimation of value functions in Q-Learning [23], Evolvable Neural Turing Machines (ENTM) for adaptive behavior in reinforcement learning [24] and One-Shot Learning with ENTM [25]. All of these techniques could benefit from the scalability of the proposed platform.

5.1.2. Distributed Evolutionary Computation (dEC)

The EC techniques can be executed in distributed processing architectures, since they have to evaluate a large population and this process can be made in parallel for each specimen. There are many ways of intercommunicating the components of a dEC process. This section describes the existing techniques for dEC and the improvements on dEC that

the proposed platform will provide. Since the quantity of techniques is very large, a framework proposed by by Yue-Jiao Gong et al. [6] is used for standardizing the description of these techniques, it splits the components of a dEC application on four layers as shown in Figure 2.

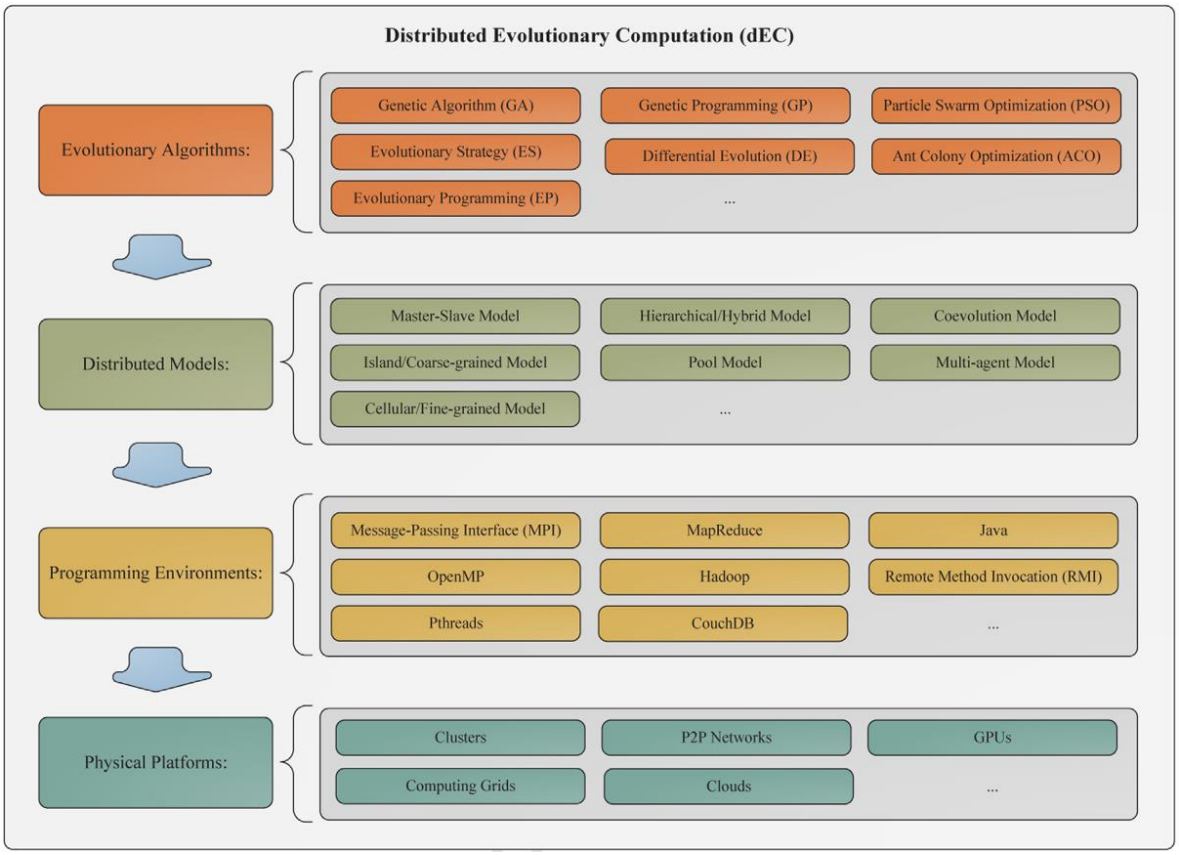

Figure 2 - Distributed EC Framework. Image source: [6]

The first layer is the *evolutionary algorithm (EA),* this is the algorithm to be executed in a distributed software platform. In the proposed platform, this layer will be user-implementable in any programing language capable of sending HTTP requests.

The second layer is the *distributed model*, it describes how the application distributes the processing load over the working nodes. These models can be divided in two sets as shown in Figure 3, the *population-distributed* models that evaluate different specimens of a population in different nodes and the *dimension-distributed* models that split the data dimensionality and evaluate each partition in a separate node. As it will be described, each model has strengths and weaknesses, but fortunately they can be mixed to overcome their limitations and leverage their strengths.

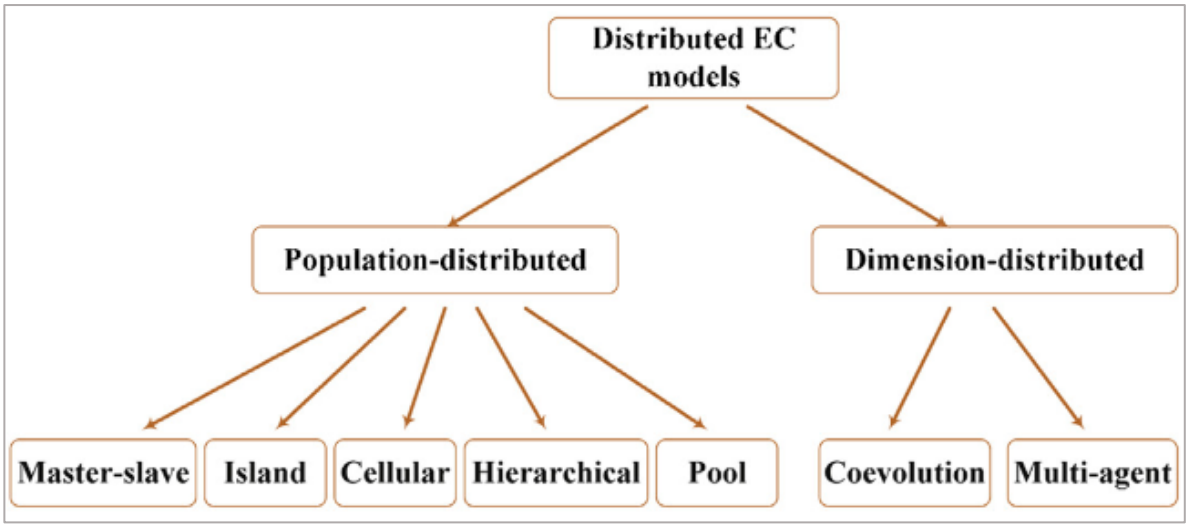

Figure 3 - Population -distributed and dimension-distributed EC models. Image source: [6]

The *master-slave* model is a centralized *population-distributed* model in which a master node distributes evaluation tasks between slave nodes. It has low *fault-tolerance* and high *communicational cost* but is easy to implement.

The *cellular* and *pool population-distributed* models divide a population in a fine-grained group structure and execute the groups in separate processors, but they also have low *fault-tolerance* since the nodes cover fixed sections of the search space. These methods can use GPU and cluster platforms very efficiently.

The *coevolution* and the *multi-agent* models are *dimension-distributed*, meaning that they divide a high dimension problem in several lower dimensional problems, but have the disadvantage of requiring a-priori knowledge on how to reassemble the results. The *coevolution* model directly assigns different parts of the input to independent populations whose results are consolidated.

The *multi-agent* model is also *dimension-distributed* and requires knowledge on how to model the EC problem as a game-theoretic game with payoffs for the players based on their actions and the actions of their immediate neighbors. The model uses an independent population for deciding the actions of each player and eventually reach a set of actions representing the Nash equilibrium, meaning that any player who deviates from it can't improve its payoff. Not all practical EC optimization problems can be solved with these two models.

Finally, the *Island model* [26] is *population-distributed,* it evaluates completely independent populations in every node and share the best specimens allowing them to exchange results with other nodes, the island model has the lowest communicational cost and the highest fault-tolerance of all the distributed models. This model also serves as higher layer of *hybrid* or *hierarchical models* composed by islands where each island

executes a master-slave, cellular or other models leveraging their advantages. The proposed platform will support Island models and hybrid models.

The third layer of the dEC framework is the programing interface, it defines the way for a node to communicate with others in order to orchestrate the distributed processing. The most common alternatives are *map-reduce* using Hadoop for java and MPI for C++ implementations, other interfaces exist but all suffer from being *programming-language* specific. The proposed platform will use a language-agnostic interface by using a REST API to provide high interoperability between heterogeneous implementations of an EA.

The fourth layer is the physical platform, some common recent choices for dEA are clusters, cloud computing and GPU computing, but they all suffer from *platform-dependency* and none of them has industrially desirable characteristics as reliability mechanisms, cyber-security or traceability. For the proposed platform, a P2P architecture has been selected since it has enhanced *fault-tolerance* compared to centralized architectures. Some implementations of dEAs on P2P networks exist, but they are limited to a specific EA, such as genetic programming [27], swarm-particle optimization [28] or neuroevolution [29].

Many dEA frameworks have been implemented, but they are either EA-specific [30], centralized [31] [32] or language-specific [33] [15] [16] [34]. None of the found frameworks has security and reliability measurements for production usage over insecure networks.

Another innovation introduced in the proposed platform is a blockchain to allow the use of trust-less nodes and to provide reliability and traceability of a dEA allowing its use in large-scale optimization projects over insecure networks.

5.1.3. Decentralized Architectural Pattern

The proposed platform features a Decentralized Architectural Pattern that allows fault-tolerance in any node participating in a distributed EC technique [35] [6]. The main difference between a decentralized or *peer-to-peer* network and a client-server network is that while in the latter a node can exclusively behave as client or server, in the former peer nodes simultaneously function as both clients and servers to the other nodes on the network and thus there are no central nodes or single points of failure.

In the decentralized architectural pattern, nodes communicate with each other forming an overlay network, commonly on top of the TCP/IP stack, but at the application layer peers are able to communicate mutually via logical overlay links, each of which corresponds to a path through the underlying network. Not all the decentralized networks use the TCP/IP stack [18], for example the *Ethernet* protocol has a decentralized architectural pattern, it is widely used in data networks, uses the MAC address of a device to communicate

directly to other devices, without intermediaries and without critical nodes for the communication with the others.

Peer-to-peer file sharing uses decentralized networking to distribute data over the Internet, the first fully P2P file sharing network was called Gnutella in 2000, it uses a simple routing scheme in which each node has a list of known nodes and relay requests to other nodes limiting the path length with a Time to Live (TTL) counter [36], this routing technique is called “flooding” and it is simple but not efficient for searching resources on the network, but as in the proposed platform the search feature is not required, the *flooding* method is used due to its simplicity.

The Bittorrent protocol designed on 2001 is more efficient for searching resources on the network by using *HTTP GET* requests to communicate with peers [18] also uses a distributed hash table protocol (DHT and Kademilia) to discover and keep track of known peers and their shared files.

In this project HTTP requests will be used for communication between nodes implementing a Representational State Transfer (REST) Web service platform to favor the interaction with a broad spectrum of programming languages and applications capable of sending HTTP requests, running in heterogeneous platforms and making the network suitable for use in low computational power devices by allowing pre-trained model evaluation with sensor data without dealing with the model training computing exigencies in these devices.

5.1.4. REST Application Programming Interface (API)

The proposed platform will feature a Representational State Transfer (REST) API to provide a language-agnostic interface to make distributed EC applications. A REST API is a way of communicating computer systems over a network. It was proposed in the Ph.D. thesis of Roy Fielding [37] and is widely used in mobile and desktop applications, especially as way of providing interoperability between devices with heterogeneous hardware capable of sending HTTP requests and receive responses from an HTTP server. The messages transmitted in the API can have any format, but the responses are text encoded in XML, HTML or JSON format. The use of a REST API is a tactic for implementing a Service Oriented Architecture(SoA) [38].

The main characteristic of a REST API is the use of the standard HTTP protocol methods such as POST, GET, PUT, PATCH, TRACE, OPTIONS, CONNECT, HEAD and DELETE on resources and items. Resources are usually data collections or processes, items commonly are collection objects or process directives to manipulate them. The REST specification does not have constraints in the type of resource or item used [37], for example: a request to obtain the first element of a collection called cars would be “GET /cars/1” and a request to obtain the set of all the elements in the cars collection would be “GET /cars/”.

The use of the standard HTTP protocol methods has additional advantages to the interoperability, such as that the HTTPS secure communications protocol can be used for serving a REST API.

5.1.5. Blockchain

A *blockchain* is a transaction ledger first used in Bitcoin and later in other applications for traceability of operations between nodes of a *peer-to-peer* network. It is a read-only distributed database running on top of a storage media for the blocks (of transactions) provided by the nodes, the storage media for the blocks can be any database or a local text file containing the block's content in JSON, XML or any other format.

Bitcoin is an electronic cash system invented by Satoshi Nakamoto in 2008 [39], it was implemented and released as open-source software in 2009. It uses a peer-to-peer network that timestamp blocks of financial transactions between accounts by hashing them into an ongoing chain of digitally signed blocks called the *blockchain.*

The *blockchain* is composed of blocks of information, in the case of Bitcoin, transactions that are cryptographically linked sequentially using a hash of the previous block to prevent future modifications to created blocks.

Once a block of transactions has been added to the blockchain, to be modified by an attacker, he would require the investment of a large computational capacity for the modification of all subsequent blocks plus its modification in all the nodes in the network for the blockchain to be valid. This provides a security advantage over a hypothetical similar system that stores the blocks without a cryptographic link. In this project, the blockchain will be used for traceability of operations between nodes participating in a dEC process.

An important part of the proposed platform is the traceability of the optimization process provided by a blockchain, meaning that the devices that discovered model parameters that increased the performance of a model are identified with their public cryptographic key and recorded in a *read-only shared database*, that is difficult to modify by attackers. The blockchain also allows the consensus between non-trusted participating nodes by collectively validating the results reported by the nodes.

The blockchain is a public ledger of information composed of signed blocks, cryptographically linked with the past ones. It has recently been used for decentralized DNS [40], voting systems [41], smart contracts [42] and private data management [43].

While in Bitcoin, nodes manifest their consensus on the validity of the transactions in a block by cryptographically signing the next generated block with a hash of a block that includes the validated one [44], in the proposed platform, nodes manifest their *consensus* on the validity of the reported performance of parameters optimized with EC by signing

the next generated block with a hash of the validated one and merging the specimens representing those parameters with their population. If the evaluation of the optimized parameters results in a different performance than the one reported, the specimens are not merged with the population and the hash of the block is not used to sign the next block.

5.1.6. Cryptographic Proof of Work

The cryptographic *proof-of-work* (CPoW) mechanism in Bitcoin has the objective of implementing a distributed timestamping service in a decentralized network, the generation of the CPoW involves solving a difficulty-variable cryptographic puzzle to create a block in a process called *Bitcoin Mining* [44]. This process requires the investment of CPU computing power, which improves the security of the blockchain since an attacker would require additional computational effort than the one used in a blockchain without proof of work to modify the block data while keeping the block valid.

The blocks in Bitcoin, are created by a hash-based *proof-of-work* of the transactions received by miners, forming a record that cannot be changed without recalculating the entire proof-of-work [39], serving as a public ledger of all bitcoin transactions allowing to calculate the quantity of currency in an account in any moment. The nodes verify the transactions in the block and express their consensus by start working in the next block using the last verified one as the last block of the blockchain. This same consensus mechanism is used in this project for validating reported results to provide reliability.

The implementation of the proof of work cryptographical puzzle is made by incrementing a nonce in the block until a hash of the block with a required number of leading zero bits is found. The computing effort required to produce a hash with leading zero bits increases exponentially with the number of zeroes and is used to provide a block time control in a network with varying number of participating miners. A disadvantage of the proof of work is that the computing power used for hashing spends a large quantity of energy and does not provide other useful result apart from the added security and the block time control.

The *proof-of-work* in the proposed platform will not be made by solving a cryptographical puzzle as in Bitcoin, but by finding new advancements in the performance of a dEC process and using the optimized parameters as *proof-of-work* providing the same block-time control mechanism than CPoW but for a useful task (optimization).

In the proposed platform, the blockchain provides traceability so a user can know the identity of the nodes that increased the performance of the model and the timeline of advancements in the optimization process. The traceability also allows to protect the process against false or corrupt reports of performance by network nodes as it contains a history of changes in performance and the reported author nodes. Also, the blockchain is

a data reliability tactic since a change in a past block of the blockchain implies the creation of all subsequent blocks and any change in a block is detectable once chained to the blockchain, protecting the information from corruption once the data has been accepted by the nodes consensus.

5.1.7. Useful Proof of Work

The main problem with the CPoW is that the computing power that it uses for mining which has no other purpose than securing the blockchain and controlling the block-time for implementing a decentralized timestamping service. There have been some attempts to provide the same functionality of a CPoW but using the computing power for a useful calculus instead of hash-based mining.

The useful calculus that is used for generating a useful proof-of-work, for implementing a decentralized timestamping service like the implemented by Bitcoin requires to be computationally hard to produce, easy to verify and to have variable difficulty.

The useful proofs-of-work known to the author solve the following mathematical problems, the verification method and the difficulty are described for each one:

- *Primecoin*: Search for chains of prime numbers, verifying that they are Cunningham chains and using the length of the chain as difficulty [45].

- *Orthogonal Vectors*: To determine if there is an orthogonal pair of vectors among *n* Boolean vectors in *d* dimensions, verifying their orthogonality and using *n* as difficulty [46], uses a pool of problems and requires one for each block.

- *3SUM*: To determine if a given set of *n* real numbers contains 3 elements that sum zero, verifying the sum of the three numbers and using *n* as difficulty [46], requires a different problem for each block, uses a pool of problems and requires one for each block.

These problems are not function of the block contents, but orthogonal vectors, 3SUM and all-pairs shortest path mitigate this problem by additionally generating a hash-based proof-of-work, like the one of Bitcoin, that additionally verifies that the hash was generated from a block containing a solution. In Primecoin, to make the proof-of-work a function of the block contents, the start of the prime chain should be divisible by the start of the hash of the block's header.

This work proposes a new useful proof-of-work that uses EC optimization problems that are hard to solve, easy to verify and have variable difficulty as described in section 6.

5.2. PROTOTYPE AND EXPERIMENT TOPICS

In the proposed experiment, an automatic foreign exchange trading agent is trained by using reinforcement learning with a distributed evolutionary computation technique being executed in the proposed platform.

5.2.1. Reinforcement Learning (RL)

As proof of concept, the proposed platform will be used to perform a distributed RL experiment in the forex trading automation domain. RL is an area of machine learning that has the objective of mapping situations to actions, maximizing a reward given to an agent that interacts with an environment by executing actions resulting from policies applied to observations of the environment and obtaining a cumulative reward [47] as shown in Figure 4.

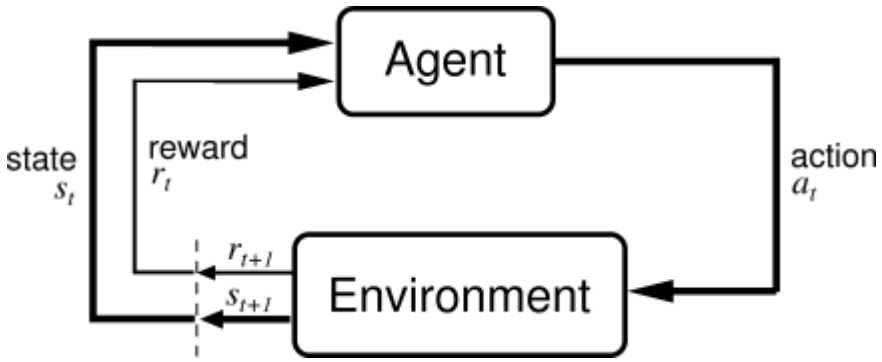

Figure 4 – The Agent-Environment Interaction in Reinforcement Learning. Image Source [47]

The agent must try the actions to learn in which situations they yield the most reward at the end [47]. This area of machine learning is different from supervised learning, because the latter is not suitable for interactive problems where desired behavior examples that are both correct and representative could be impractical to obtain. A common feature between the reinforcement learning problems is that the agent can use its own experience to improve its performance in the long term.

The four main components of a reinforcement learning system are: policy, reward function, value function and optionally an environmental model. The *policy* is a mapping of the observations made by the agent into actions that must execute in the observed situations, the *reward* function is the immediate goal of the agent due to an action executed in a situation, meaning that the reward function maps a state-action pair to a numerical value called reward, the objective of the agent is maximize the total reward it receives in the long term.

Rewards are given directly by the environment, but values must be estimated from the sequences of observations an agent makes over its lifetime. The *value* function indicates the total amount of reward that an agent can expect to accumulate starting in some state.

It represents the desired behavior of the agent in the long run. The fourth component of reinforcement learning is a mathematical model of the environment in which the agent will act with the objective of predicting a state given an action with the purpose of planning.

Some reinforcement learning methods are aimed to estimate *value* functions, these methods are more suitable to problems with delayed rewards, other methods search directly in the policy space and are more suitable for environments where rewards are immediate. Both methods have been used with EC [48] [23] and since the trading automation problem has both delayed and immediate rewards, both methods will be trained simultaneously to validate the multi-processing capability of the proposed platform and to compare their performance.

There are three threads in the history of reinforcement learning, the optimal control, trial-and-error learning and temporal-difference.

The first thread is the optimal control, that was initially defined by Richard Bellman in 1957. He proposed the general problem of designing a controller to minimize a measurement of a dynamical system's behavior over time, to solve this problem he uses an "optimal return function" called the Bellman equation which is a Markovian Decision Process (MDP) and must be solved with dynamic programming, which is considered the only way to solve general stochastic optimal control problems, but suffers from the curse of dimensionality since its computational requirements grow exponentially with the number of state variables. These methods are not used in this work.

The second thread which is used in this project, is the trial-and error learning, detailed in the previous section (Evolutionary Computation). The third thread, the temporal difference, refers to the methods that use successive estimates of the possible actions and their rewards for selecting the best next action.

Chris Watkins brought together the temporal-difference and optimal control threads in the Q-Learning technique in 1989 [49]. Several evolutionary computation techniques are also applied in the trial-and-error techniques like Neuroevolution for estimation of value functions in Q-Learning [23] [50] [51] [52] and Evolvable Neural Turing Machines for adaptive behavior learning [24].

An important advancement in reinforcement learning is the ESP technique *(Encapsulation, Syllabus, Pandemonium)* [53] that uses evolutionary computation to learn tasks requiring complex behavior of an agent, dividing the main task in human-defined sub-tasks that can help to the reusability, modularity and hierarchical composability of learned behaviors.

There exist some software frameworks with reinforcement learning environments as RL-Glue, RLPy or The Arcade Learning Environment, but the newest and with the more

standardized way to test and measure the performance of reinforcement learning algorithms is the *Open AI Gym toolkit* [54] that enables running an agent that interacts with an environment with a standardized software interface in Python allowing adequate experimentation with the controller implemented in a different programming language from the used in the environment and allows to extend its functionality to custom environments as the one we use in this work [55].

The applications of RL include control, decision making and optimization tasks, several records of performance in have been established by modern reinforcement learning programs, in some cases defeating the human counterparts, for example Google Alpha GO [56] or Atari [57] among others.

5.2.2. Automated Foreign Exchange Trading

As a prototype application of the proposed platform, an automated trading agent in the forex market was implemented. The foreign exchange market is the largest market in the world, it is also called Forex and is a global, decentralized market for currency trading, works 24 hours, every day of the year except for weekends.

The international banks and financial institutions around the world make transactions and report them as legal currency exchanges, each actor estimate the exchange rate for future transactions based on the current rate, the volume of transactions and the demand of both currencies in the market.

The factors driving the demand of each currency have both short and long-term causes, making the market susceptible to perturbation in the short-term by speculation and technical factors like trends or volatility, while in the long-term fundamental factors as interest rates changes, increments in industrial production or trade alliances are the driving force behind the change in a currencies demand.

To illustrate the difference between the goods and services trading and the foreign exchange trading markets, the total exports of United States in 2013 were $2.26 Trillion **USD/year** [58] compared with $5.3 Trillion **USD/day** traded in the Forex market in average in April of the same year [59].

A Forex broker is a financial institution that exchanges currency, but it is differentiated from a typical currency exchanger in that it has accounts and some features that allow automated or manual buying and selling of batches of a currency paying with another one. A typical trader invests some *initial capital* and load it onto his broker account, then he can discretionally open a buy order of a batch of currency when the exchange rate is low, and close it (sell the batch) when the exchange rate arises, keeping a profit if the market moves as expected or losing a quantity otherwise [60].

The quantity of capital a trader has in his account plus the profit (or loss) of the current open orders is called the *equity* and is the quantity a trader would have if he closes all the orders he has opened at any moment. *Equity* is also the money he would have available to open new orders**.** When the equity is less than zero, it is said that a *margin-call* event occurs [61] and the broker automatically closes the orders and discounts the currency from the trader's account to avoid losing funds itself. In a *margin-call* the trader loses all of his funds, so this is the main situation to avoid for a trader. The *equity* is the quantity the trader wants to maximize with an acceptable risk, by using small order sizes in proportion to the equity to reduce the risk of *margin-call*.

There are other important quantities provided by the market that the trader must consider before opening or closing an order, the first is the exchange rate *spread* [62]**,** it is the difference between the buying and selling price of a currency, it represents the earnings of the broker and commonly are set dynamically based on the volume of transactions and the volatility of the market calculated as the standard deviation of price in the last term.

The *pip* is the minimum unit of variation of an exchange rate, i.e. 1pip=0.00001 for EUR/USD. The minimum *spread* is commonly between 5 and 20 pips and the maximum between 50 and 100 pips, depending on the broker and the type of account used, for example, if the nominal EUR/USD rate is 1.954877, when the trader buys on low volatility with a spread of 10 pip, the order is executed at 1.9548**87**, while if the spread is 20 pips, the order is executed at 1.9548**97**.

Finally, the *leverage* [63] is the proportion of the traders currency that his Broker is allowed to borrow him automatically, commonly this leverages are from 1:50 to 1:400 depending on the broker and account type, meaning that when he opens an order, he can buy batches of currency of 50 to 400 times the quantity of money he used for the order, but when he closes the order, the money is returned to the broker letting a larger profit or loss per pip variation for the trader than if the operation was executed with 1:1 leverage. As general rule for most brokers, larger leverages also come with larger spreads.

The main task of a trader is to select the most appropriate moment to open or close orders trying to maximize his equity given some market conditions while keeping the order size small thus reducing the risk of a margin-call, additionally the trader must select the best currency to open an order at a given time, since volatility may vary between currencies at the same time of the day and investing in one currency can be less risky than investing in other one, but these are commonly experience-acquired skills.

Automated trading agents has been described by literature [64], and can interact with the Forex market by using a software provided by a financial institution, commonly a dedicated Forex broker. A common software used for trading is the Metatrader platform with its programming language MQL, that can communicate with external programs via

HTTP requests to send market values and has full automatable control over the actions on an account. Care must be taken before opening an account with a broker and verify that is a well-credited and officially registered Broker in its country of origin. For testing purposes, Metatrader can be used in Demo-Mode in most brokers.

Training an automated trading agent with reinforcement learning requires a model of the market, more precisely a model of a broker account that interacts with historical market data [65], so the agent's performance can be measured on this model and later with a trained model, a program can be implemented in any language to communicate with a Metatrader application to get the model inputs and to control with its outputs a real live or demo account.

The data available to a trading agent [66] is composed of multiple time-series, some of the available ones are the exchange rates, spreads and volume of transactions of multiple currency pairs, technical indicators calculated on the exchange rates and fundamental indicators as the Nasdaq, S&P, oil price or gold price, but also predictions from machine learning techniques may be used to identify patterns in the time series by generating a new time series with the forecast and using it as part of the input of the agent being trained.

6. OPTIMIZATION-BASED PROOF-OF-WORK

In a decentralized network, participating devices may have different clock configurations because there is no central server that decide the timing of the events of interest. Because of this, in Bitcoin Satoshi implemented a decentralized timestamping service using a hash-based cryptographic proof-of-work (CPoW) that timestamps block of transactions, reaching consensus on the order of transactions in a decentralized network to avoid the double-spending problem.

6.1. PROOF-OF-WORK SECURITY

The decentralized timestamping server implemented in Bitcoin can be difficult to adapt to other applications since requires a large computational capacity for producing the CPoW. The purpose of the Bitcoin's CPoW is dual, first it serves to control the block time (period between consecutive blocks) since it has variable difficulty and second, it serves to protect the blockchain from being changed since attackers need to spend a large computational capacity to make a change and generate a valid blockchain with it.

This kind of security is required for an application such as Bitcoin where financial transactions are the contents of the block and the purpose of the whole network is to secure them, but for a non-financial application, this security requirement can be of less priority. The problem with the Bitcoin's CPoW is that the computations for producing it does do not make any useful work other than hashing the block contents.

In a non-financial application, without the same security requirements, it may be desirable to use a decentralized timestamping server and to use the advantages of decentralized networking and traceability but making useful calculus with its CPU capacity instead of repetitive hashing for producing a CPoW.

6.2. USEFUL PROOF-OF-WORK

As mentioned in section 5.1.7, there are useful proofs-of-work that control the block-time by solving problems such as orthogonal vectors and 3SUM [46] and they have variable difficulty. These problems are not function of the block contents, but they mitigate this difficulty by generating a hash-based proof-of-work, like the one of Bitcoin, that additionally verifies that the hash was generated from a block containing a solution.

This document proposes an Optimization Proof-of-Work (OPoW) to control the block-time by solving an optimization problem in a decentralized architecture. In a dEC optimization process in a decentralized architecture, the problem of possible different clock settings due to absence of central servers exist and causes that an optimizer can't be sure if it is using the latest optimization state since older ones may appear newer to other optimizers with different clock settings.

Another problem is that in such hypothetical decentralized dEC optimization architecture, due to the same problem of different clock settings, any traceability made to events in the network will be unreliable since the timing and order of the events cannot be known accurately.

6.3. OPTIMIZATION PROOF-OF-WORK

A proof-of-work is defined in the in the Bitcoin paper [44], its main characteristics are:

- The PoW should be function of the contents of the block for allowing detection of changes in the block after its creation to protect blocks against modifications.
- The PoW should be hard to produce but easy to verify.
- The PoW should have variable difficulty for controlling the block-time with a varying number of miners.

This project proposes the use of an optimization-based proof-of-work (OPoW) for controlling the block-time for a blockchain, it is hard to produce, easy to verify and it has variable difficulty for controlling the block-time, but it is not function of the block contents and thus it has a security tradeoff relative to the Bitcoin's CPoW since it doesn't protect the blocks from being modified after their creation.

The difficulty in the Bitcoin's PoW is the number of zero bits at the start of a hash of the block contents, in the proposed OPoW the difficulty is a threshold of performance that must be reached by the optimization of the parameters of some mathematical model using distributed evolutionary algorithms.

The OPoW itself is a hash of the block contents that must contain the latest optimization state represented by model parameters, an external user can verify the block by generating the hash of the block contents and verifying that they contain parameters that after being evaluated with some validation data result in a performance measurement as a real number.

The OPoW is used in the blockchain in a similar way to the Bitcoin's CPOW, every block must contain a OPoW of the previous block (hash of previous block containing new optima), but with the difference that in case of network fragmentation in Bitcoin the longest chain is selected as valid since it represents the highest computational effort invested , but if using OPoW the chain that has the highest final performance is selected as valid for the same reason and all the information on the fragmented chains is flooded to be included in the next block of the valid chain.

6.4. BLOCK-TIME CONTROL

The block-time determines the time between consecutive blocks, depending on the application, it may be desirable to have a constant block time or not. There are two options for block-time control using OPoW:

1. To use a minimum increment in performance of an optimization problem as threshold for controlling block creation.
2. Not to use a block-time control system at all, generating a block with every performance increment, with the result of a variable block time.

For controlling the block time in Bitcoin, the next difficulty is calculated from a moving average targeting a desired block time and a similar method can be used in the proposed OPoW but for calculating the next performance threshold that an optimizer must reach to generate a block.

There are also two options to use OPoW with block-time control using an increment in performance as difficulty:

1. Using an increment in performance as difficulty and calculating the next difficulty as a function of the desired block time, the previous difficulty and previous block time.
2. Using both the increment in performance and the leading number of zeroes as a composite difficulty in a way similar to [46], that uses an hybrid approach requiring an increment in performance, but additionally requiring that the OPoW has a number of leading zero bits determined by the difficulty to achieve a desired block time and additionally mitigating the problem of the OPoW not being function of the block contents with a threshold between security and usefulness.

In the second option, the tradeoff between security and usability hypothetically could be controlled by having the optimizer to produce both an OPoW by generating an optimization state with a performance superior to the block-creation threshold and a CPoW similar to the used in Bitcoin with its independent difficulty consisting of the number of leading zeroes of a hash of a block containing both the previous block hash and the OPoW that is a hash of the block contents that must contain the last optimum report request from an optimizer. This approach will allow to have the security provided by a CPoW, at the expense of a reduced difficulty, in exchange of the usability of the OPoW for optimization.

6.5. EMPIRICAL VALIDATION

For empirically validating the scalability, fault-tolerance and invalid result rejection provided by a decentralized architectural pattern using OPoW, a software platform for decentralized optimization was designed and implemented. Also, an optimization application was developed for the mentioned platform to perform the following 3 experiments to validate the expected behavior of the OPoW:

1. **Scalability:** By adding nodes to the network, their combined computational capability is incremented.

2. **Fault-tolerance:** The removal of any of the nodes of the network must not stop the application from working.
3. **Invalid result rejection:** The attempt to report an invalid result during the generation of an OPoW must be rejected by participating nodes.

These empirical validation tests are described on section 9, the software platform design process is described on section 7 and the technical decisions for the implementation are described in section 8.

7. PLATFORM DESIGN

The design process for the software platform using the proposed optimization proof-of-work is described in this section. This process is based on the Attribute-Driven Design (ADD) iterative design methodology described in [38]. The platform design process had the following steps:

1. Define a set of architecturally significant requirements (Section 8.1).
2. Identify the elements that compose the desired system, their behaviors and their interactions (Section 8.2).
3. For each of the elements identified (Section 8.3):
   a) Identify all the requirement associated with the element
   b) Generate a design solution for the selected element:
      i) Select the architectural pattern(AP) that fits the best to its requirement
      ii) Select the tactics that are more suitable for implementing the selected AP
      iii) Select the technologies used for implementing the selected tactic

7.1. REQUIREMENTS

This section describes the stakeholders, their business goals and their functional and non-functional architecturally significant requirements extracted from stakeholder business goals and a hypothetical user story.

7.1.1. Stakeholders

For the proposed software platform, the stakeholders are the developers that make applications in the platform, and the users that consume the applications that developers make, their goals are useful to identify the requirements. The stakeholders have the following business goals:

- *The developers* create applications that use the results of optimization of some model using EC, they benefit from making these optimization processes scalable secure and reliable by:
  - Extending existing evolutionary algorithms into a distributed architecture by sending an HTTP request between iterations to receive the best genomes and update their populations with them *(migration operator*).
  - Using the most optimized parameters while the mathematical model is being optimized on an existing EC process.
  - Creating decentralized applications with traceability of some variable.
- *The users* benefit from consuming software that uses optimization with EC, they benefit from using the latest optimized parameters for some model but also from having the capacity to collaborate on large optimization processes whether for free (i.e. an academic experiment) or being by their invested computational capacity

paid by using traceability information, they can have the following interactions with an application built in the platform:

- Consume services offered by an application.
- Contribute their compute capacity to an application for scaling the computing capacity of a network working in an EC process. The blockchain allows to keep a ledger of activity that can be used to bill for services.
- Provide third-party services that use the current state of an EC process.
- Consume services provided by a platform-built app with their operations logged in a distributed read-only database (blockchain).

7.1.2. User Story: Forex Trading Automation

The following user story was used to identify the requirements and the desired quality attributes, in this story, the developer deploys an application for distributed optimization of models' parameters and remote evaluation of models with client inputs, 4 different roles will be used in this scenario:

- *Client*: Sends inputs to be remotely evaluated by the model with optimized parameters.
- *Evaluator*: Produces an output given a mathematical model, its parameters and some input.
- *Optimizer*: Implements an EA for optimizing model's parameters.
- *Node*: Orchestrates the communications between devices and other nodes, also implements traceability of actions performed by devices in the network. The nodes must have some form of authentication and authorization for controlling the access to the network.

A financial organization wants to establish a market around automation of forex trading. The purpose of their business is to create a market where people worldwide can buy automation signals and sell their computational capability to help optimizing a neural network in a reinforcement learning task for automated forex trading. The clients send their market information and account status and to an evaluator and the trained network should return the following action with the best possibility to obtain profit. For this purpose, the organization commissions a developer to make an application that has the following capabilities:

Optimizers: A group of optimizers execute an evolutionary algorithm for optimizing the parameters of neural network (both topology and weights), they all share their optimization state between iterations of the EA algorithm. When a new optimum is found, they synchronize their optimization state with other optimizers, so the others can use the latest optimization state in the next iteration of their evolutionary algorithm. When they report an optimum, it is recorded in a read-only database for further possible payment to the respective optimizers for the efficiency increment they achieved. The optimizers also

can validate reported states' performance, report invalid results and use the state of the best performer in the next iteration of their evolutionary algorithm.

Evaluators: A group of devices that evaluate inputs of a remote client (market information and order status) with the best performing model parameters. They send as response the output of the evaluated model as the next action the client should perform. The responses of the evaluators are recorded for future billing for their services.

Clients: The clients can perform the following activities:

- They can request evaluation of an arbitrary input, (for example, market information and order status) in the best performing model and obtain the output of the model (neural network) that can be used by the client as the next action.
- While the distributed optimization is happening, the clients can monitor from their cell phones or other Internet-enabled devices, a real-time plot of the performance of the optimization process.
- They can download the best performing model at any time.
- They can view the event log of the distributed process containing timestamped activities of all participants.

Nodes: A group of devices act as nodes, they can be in any location worldwide and their job is to receive the requests from all other participants, their functions are:

- The clients, evaluators and optimizers can send their request to any of the available nodes.
- They all store a list of known nodes.
- They implement authentication so unknown devices cannot participate in an optimization process.
- They implement authorization, so the devices can perform only their assigned role.
- The nodes must send updates of the events happening to all other nodes.
- They store the best parameters found so far by the optimization process.
- The historic data such as event logs and optimization states must be very difficult to modify once produced and the historic data is shared among all nodes participating in a process.

When the experiment is being executed, the clients can request evaluations for automating their forex-trading systems, optionally paying for these evaluations using the traceability data stored on the nodes as a record of services consumed. The optimizers can also bill for their invested computational capacity in optimizing the model's parameters.

7.1.3. Non-Functional Requirements

The non-functional requirements are useful to determine the quality attributes that the platform must have. The proposed platform is intended to be used for the development of applications that make use of distributed EC processes with the following requirements:

- Scalability: The capacity that new optimizers can be added to an existing optimization process, providing additional computing capability to a dEC process.
- Fault-tolerance: An existing dEC process must continue to execute in the event of failure of any of the nodes participating in it.
- Traceability: All the operations between devices such as creating a user, reporting new optimal parameters of requesting an evaluation must be logged on to a ledger so historic data can be reviewed for debugging a dEC process and for accounting of services consumed and provided by participating devices.
- Interoperability: The platform must provide a method for using heterogeneous implementations of the devices' roles in multiple programming-languages.

7.1.4. Functional Requirements

The functional requirements are useful to identify the desired behavior of the system or its reactions in a runtime environment. The following ones were observed in the shown use case:

- The proposed software platform must allow the development of applications for optimization with distributed evolutionary computation using existing implementations of evolutionary algorithms by synchronizing optimization states using a request to a node.
- The **nodes** store a log of interactions with other devices.
- The **nodes** also store a ledger of the optimization state (model parameters), this ledger must serve for storing the actions performed by other devices.
- An authentication, authorization and accounting framework must be implemented for defining the roles of devices that connect to **nodes**.
- All authorized devices have access to a Web interface for monitoring the performance of the models being optimized.
- The users that own a **node** are called administrators and have the capability to create new EC processes and to give permission for certain users to connect new optimizing nodes.
- Multiple **nodes** must synchronize so any device can make requests to any nodes they can reach.
- The way in which an **optimizer** interacts with the platform is by sending requests to a node for requesting the last optimization state or for posting a new optimum that has been found. Optimizers can independently validate the states.
- The way in which a **client** interacts with the platform is by sending requests to a *node* for requesting an evaluation or consulting the results of an evaluation.

- The way in which an **evaluator** interacts with the platform is by sending requests to a node for requesting data and model parameters for performing a *pending evaluation* or for posting the results of an evaluation.

7.2. COMPONENTS AND BEHAVIOURS

In this section, the components of the desired system are identified. These components and their desired behaviors are:

- **Client**: Requests evaluations of client's data with a pre-trained model.
- **Evaluator**: Requests optimized model's parameters and performs evaluations
- **Optimizer**: Requests optimized models, perform optimization, report optimum found.
- **Node**: Handle the requests from other devices, logs variables, store updated model parameters, and communicate with other nodes using a decentralized architectural pattern, implements traceability of interactions with other components using a blockchain and OPoW.

The nodes use a decentralized architectural pattern, meaning that an optimizer, evaluator or client can make requests to *any* of the available nodes in contrast to a centralized architectural pattern where devices must make requests to just *one* node as shown in Figure 5. The proposed platform also behaves as centralized if only one node is used.

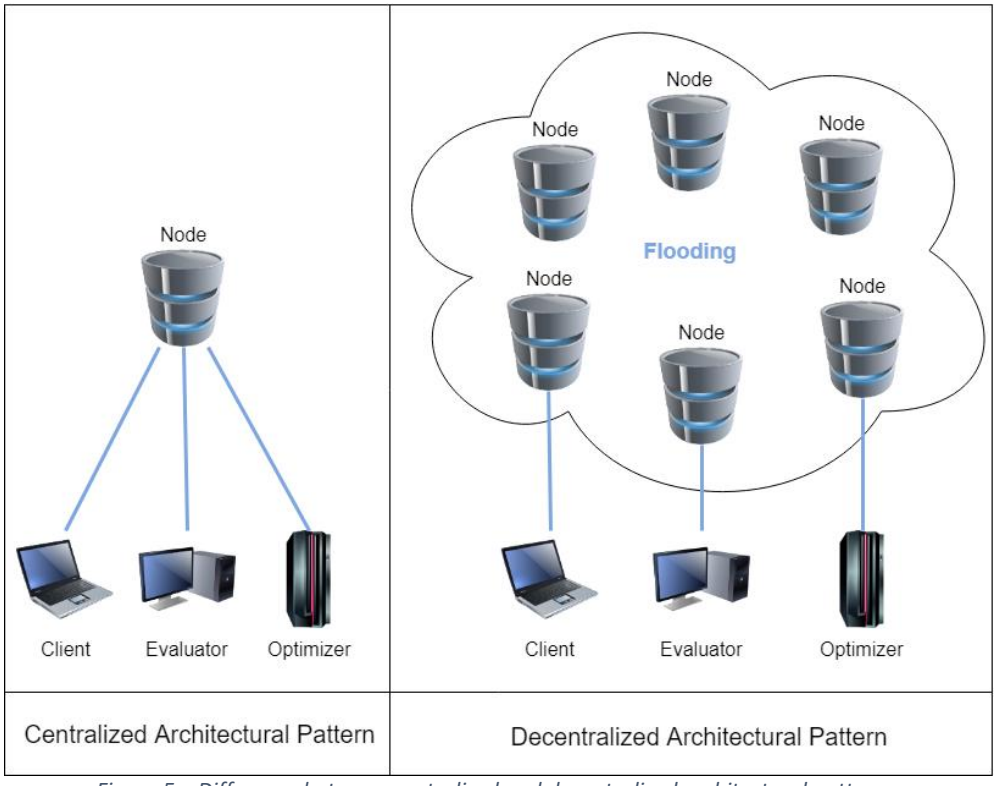

Figure 5 – Difference between centralized and decentralized architectural patterns.

A diagram of the platform with all the components and their expected behavior is shown in Figure 6. In step 1, the client sends an evaluation request to any node, the node save the request as a register in the evaluations table as a pending evaluation and flood the request to the other nodes, so they save the same register of the pending evaluation, the client then starts to periodically perform step 5 to verify if an evaluation has been performed, and to retrieve its results. In step 2, an evaluator retrieves the list of pending evaluations from any node and selects one to evaluate, in step 3 the evaluator retrieves the selected evaluation data and the last parameters of the model that the evaluation requires. After evaluating the data in the model, the results are sent to any node in step 4.

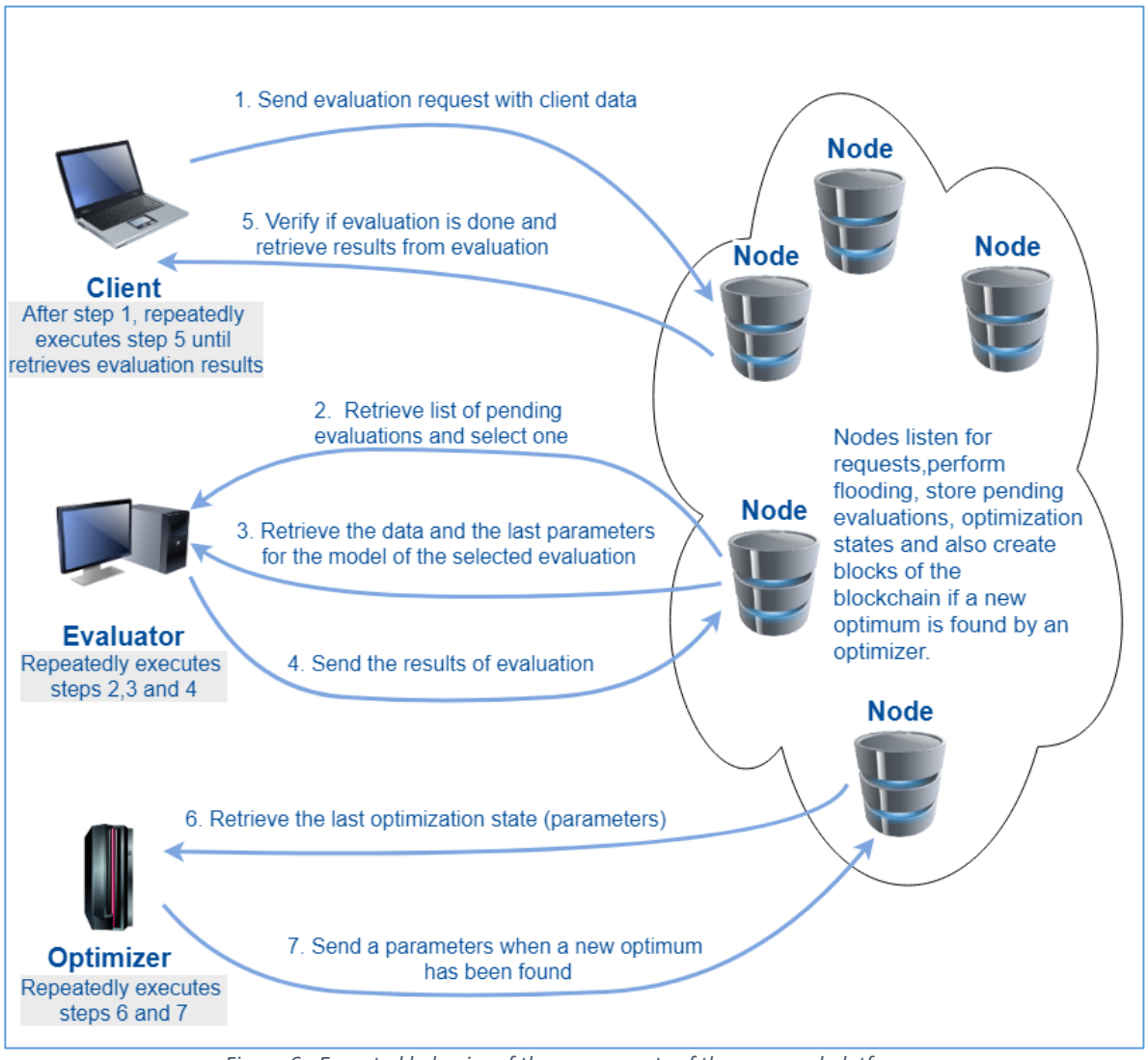

Figure 6 - Expected behavior of the components of the proposed platform.

Steps 6 and 7 are repeatedly executed by the optimizers to retrieve the last optimization state between iterations of the EA algorithm and to send its optimization state to the node if a new optimum has been found. Nodes are constantly listening for requests,

flooding received requests, storing pending evaluations, optimization states (model parameters) and creating blocks of the blockchain if new optimum is found and the block creation conditions are met.

The benefits of using a decentralized architectural pattern are fault-tolerance and scalability, but in a peer-to-peer network there exists the problem of consensus on the order of events since nodes can have different time configurations and packets can take variable times to reach a destination.

To solve the problem of consensus on the order of events in a decentralized network, nodes implement a distributed timestamping service like the used in Bitcoin, using blockchain and OPoW for allowing traceability in the node's decentralized architecture.

Each component need access to Internet or a LAN for communicating with other components, every component is a program that perform the desired behavior. More than one component can be used simultaneously in the same computer, virtual machine or even the same program as in the case of an *optimizer* that can also perform evaluations between iterations of the *EA*.

7.2.1. Usage of Nodes

If a developer wants to deploy a network of nodes, the proposed platform provides the developer with an already working node, the developer must configure the following using the Web interface of the node or a text editor:

- Configure the network topology by configuring each node neighbors IP addresses.
- Create the usernames and passwords of the users (other nodes, optimizers, evaluators or clients), these will be used in each request to any node.
- Create one optimization process, multiple processes with unique identification (*process_id*) can be created in a node. The node stores one separated ledger of events for each process. The user can do this using the Processes menu in the Web interface. The *new process_id* must be included as a parameter in all the REST API requests made by evaluators, optimizers and clients to differentiate disctinct optimization processes that may be using the same node.
- Authorize each username for interacting with the node in a specific *process_id*, using the preferred component role, for example, if a user is authorized as *evaluator*, he cannot report new optimization states unless he is also authorized as *optimizer*.
- After completing the previous steps, the developer must set the flag *active* to *true* for the configured process using the Web interface or the command line, so the nodes starts synchronizing with other nodes and listening requests of the different components that participate in the optimization process. The node only store information about the optimization state but does not save the datasets used for training or validation, these datasets must be stored in each optimizer. The

platform provides a field for the hash of the training and validation datasets, for the optimizers to verify their datasets before using them.

The nodes synchronize their states and share with other nodes the requests that they receive by flooding. The work of the developer after having a network of nodes active is to setup the optimizers, evaluators and clients that will connect to any of the nodes to perform their tasks.

7.2.2. Usage of Optimizers

If a developer wants to implement an *optimizer* in the proposed platform, he must have an already implemented *EA* working on a computer and he must modify the code of the *EA* to add the synchronization of its optimization state with any node between iterations. In multitasking operating systems such as Linux, the same computer can be used to execute a *node* and simultaneously an *optimizer* as separated program. An *optimizer* also can be used simultaneously with an *evaluator*.

7.2.3. Usage of Evaluators

If a developer wants to implement an *evaluator*, he can use a computer for periodically requesting pending evaluations from any node, performing them and report the results to any node or he can use the same *optimizer* program, updating the code so the *optimizer* not only synchronizes its optimization state, but it also performs pending evaluations between *EA* iterations. In this case the same program is the optimizer and the evaluator.

7.2.4. Usage of Clients

If a developer wants to implement a *client*, he can use an independent computer for requesting to any node remote evaluations of some input on a model being optimized (e.g. neural network). Alternatively, the developer can use the client simultaneously with other components such optimizers, evaluators or nodes on the same multitasking OS depending on the requirements of the application being developed.

7.3. DESIGN OF IDENTIFIED COMPONENTS

In this section, the design of each component is presented. For each component an architectural pattern is selected with a tactic for implementation and the technologies that will be used for their implementation.

7.3.1. Node

The node listens for requests from other nodes, and from clients, evaluators and optimizers, as described in sections 7.2, 7.3.2, 7.3.3 and 8.3.4. The nodes keep a register of the requests made by other devices for traceability and orchestrate the communication between them. In this section, the architectural patterns used are described.

7.3.1.1. Architectural Pattern

For the requirements defined in section 7.1 that deal with interaction between nodes, optimizers, evaluators and clients, the *SoA* pattern was selected since the optimizers, evaluators and clients require to perform remote requests to the nodes.

However, for interaction among nodes, the architectural pattern that fits the non-functional requirements of scalability, fault-tolerance, traceability, information reliability and availability is a *decentralized architectural pattern* because the other alternative, the SoA architectural pattern, does not provide fault-tolerance and has very limited scalability and availability. Thus, the more suitable pattern combination for the whole project is *decentralized architectural pattern* for node to node communications and *SoA* for the communications between each node and clients, evaluators and optimizers as shown in Figure 5.

Optimization State Synchronization in a Decentralized Architecture

The optimization state of a dEC process is a population of genomes, where each genome represents a set of mathematical model parameters such as neural network weights. The synchronization of optimization states between different optimizers can be made using a migration operation that transfer one or more of the best representatives of a population to other optimizers, replacing the worst performing genomes from them. In this way the newly synchronized optimizers use the best performing genomes.

Traceability allows nodes to synchronize their optimization state by using the last one reported in the trace of operations (blockchain) in the decentralized application. As a tactic to achieve traceability in this decentralized architecture, since the time of the participating nodes is not synchronized because there is not a central node, a decentralized timestamping service was implemented as described in section 6 using blockchain and OPoW.

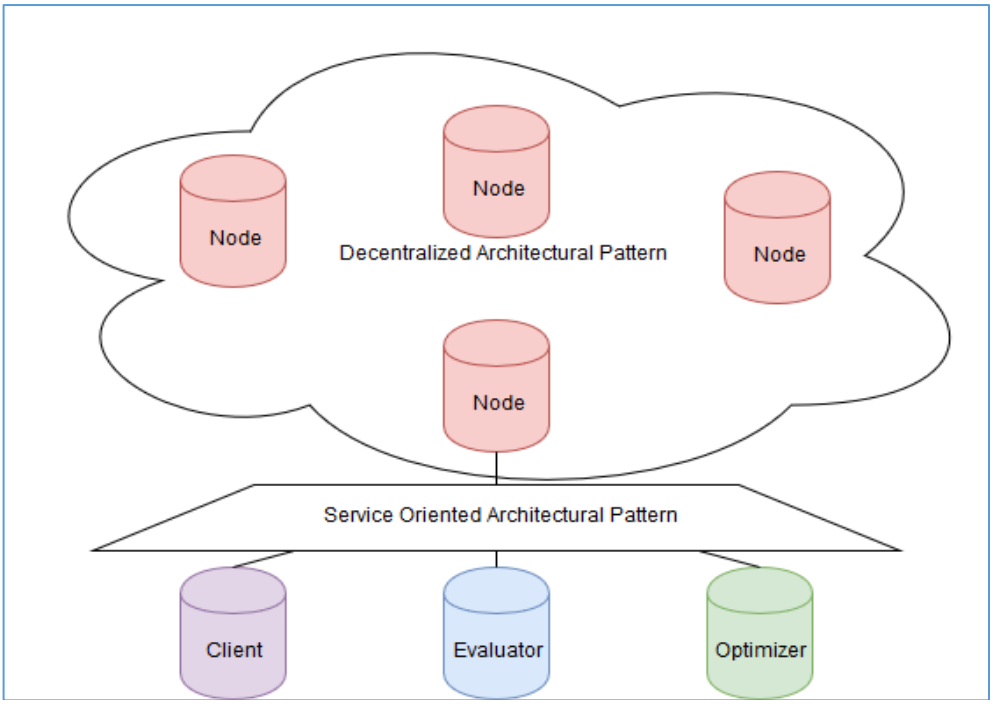

Figure 7 - Platform Architecture

7.3.1.2. Suitable Tactics

As mentioned in the justification in section 4, the interest in tracing the events (requests and responses) during dEC in the proposed platform has two motivations, the first one is that it allows the developers to debug a decentralized application that uses dEC and allows the developers to have an historical account of how the participating devices contributed to a decentralized optimization process.

The second motivation to have interest in implement traceability in a decentralized optimization application is that it allows to have an historic log of operations performed by the clients, evaluators and optimizers that can be used to bill and pay for the services consumed and provided by devices in the network, since the clients consume the evaluation services provided by evaluators and evaluators consume services provided by the optimizers since the evaluators use optimized model parameters.

For implementing the *decentralized architectural pattern*, and to satisfy the requirements of traceability in a decentralized architecture, a blockchain is the only suitable tactic as far as we know. Also, for implementing the *decentralized architectural pattern*, and to satisfy the availability, scalability and fault-tolerance requirements, *flooding* is the only suitable tactic as far as we know.

For implementing *traceability* over a decentralized network and overcome the problem that there is no central server to synchronize nodes' times, a decentralized timestamping service was implemented using a blockchain with OPoW.

Also, a ***Blockchain*** can be used as a mechanism for storing read-only information in a decentralized database. It does make detectable any change made in previously stored information, by signing every new block of information with a hash of the previous one, forming a chain that can be verified to assess if the information has been modified since it was initially stored. This allows to keep a register of operations (evaluation requests, optimizations) that can be used to bill for services provided or consumed or for any purpose the developer requires, such as debugging or security event logging. This also allows *trustless collaboration*, since the optimizers that report the optimized parameters do not need to be trusted because their results can be independently verified by any other optimizer and attempt to introduce an erroneous result in the network is futile since no optimizer will use it if it is invalid.

The **blocks** are registers in a table (*blocks table*) in a relational database, and the contents of each block are registers of another table (*accounting table*) that contain as registers the requests and responses received and sent by the nodes, each one belonging to a block in the blocks table. An important request that is saved in the blocks is the *CreateParameter()* that optimizers send to the node when they find a new optimum, the node verifies if the parameter met the block creation conditions and create a block if so, broadcasting the new block for other nodes to update their accounting registers as belonging to the new block by setting the field block_id to the identifier of the new block in the blocks table. But also other requests are saved in the blockchain such as the *CreateUser()* request that is used to create new users with their passwords and roles (clients, evaluators, optimizers or nodes).

Flooding is a tactic for broadcasting a message to all or a group of available peers in a network. Flooding is an algorithm in which every message arriving to a node is sent to every known link except the one it arrived on. The known links are saved in a local database in each node.

There are two types of flooding available: uncontrolled and controlled flooding.

In the **uncontrolled flooding** of messages, nodes forward incoming messages to all known peers except the message source, but if the network has cycles, the packages will loop indefinitely and eventually produce an overload of traffic causing a denial of service (DoS).

In the **controlled flooding** of messages, nodes forward the messages according to some control mechanism. The most basic mechanism used for controlled flooding is the use of **Time to Live (TTL)** in which a message is sent with an additional field containing a counter and every node that receives the message, decrease the counter and do not forward the

message when the counter reaches zero. This prevents inifite loops but causes a temporary increase in the network throughput, that may not be desirable. This mechanism was used in this project.

Other mechanisms for controlled flooding exist for reducing the loops such as the Reverse Path Forwarding (RPF) that requires to remember the routes from any node to every other or Spanning Tree Protocol (STP) that requires the creation a graph of routes in the network with no loops (a spanning tree).

Due to the time requirements of this project, the most suitable algorithm for controlled flooding is the **Sequence Number Controlled Flooding (SNCF)**, this algorithm adds a field to the message for the address of the sender and a sequence number to identify the message (or a hash of the message) and nodes only forward messages that it has never seen before, in this algorithm nodes can receive a message more than once but will only forward each packet once. Both SNCF and TTL were used in the proposed platform.

7.3.1.3. Selected Technology

The selected technology is REST because its use of standard HTTP methods, simplicity of implementation in mainstream programming languages and descriptiveness of methods on the management of data collections, additionally, it does not enforce the use of JSON in the request parameters and allows both XML and JSON based responses that can be formatted in standard specifications such as SOAP v1.1 [69] or JSON-RPC v2.0 [70]. JSON-RPC v2.0 is used in this project due to the lower traffic overhead caused by the simpler header and footer structures. A RESTful Application Programming Interface (API) was implemented in the nodes allowing interaction from the optimizers, evaluators and clients via HTTP requests.

7.3.1.4. Implementation

As the optimizers require to perform HTTP requests for both asking for the last optimization state and for posting a new optimum, the node must use a table to store the parameters, also it requires a table to store the pending evaluations. The nodes create a new block when one node receives a new optimum and it meets the conditions for new block creation (performance is superior to certain threshold).

The three methods identified on the node are *GetProcess()* accessed by the route *GET /processes/process_id* that returns *parameter_id,* the identificatory of the most optimized parameters of a process and *GetParameter()* accessed by the route *GET/ parameters/parameter_id* that returns an optimization state (a set of optimized model parameters) with identification *parameter_id* in the table parameters. The third method identified is *CreateParameter()* accessed by the route *POST /parameters* that allows to create a new register in the *parameters* table, and automatically create a new register in the blocks table if the reported *performance* in the parameters is higher than the current threshold.

Contents of a Block

Each register in the blocks table represents a block of a blockchain. The *blocks* table contains fields for unique block identification, optimization process id, performance threshold that future blocks need to reach, contents (composed of a list of identifications of registers of the *accounting* table), a SHA-256 hash of the concatenation of the registers of the *accounting* table that belong to a block and the previous block hash and a field for the optimizer username that created the new optimum.

The *accounting* table represents the contents of a block, it records all the requests made to the node including flooding requests with fields for its route, method, request parameters and response, this table also has a field for the block it belongs (*block_belonging*) in the blockchain. Every time the node receives a request, a new *accounting* register is created and after the corresponding method has been executed locally, the node floods the accounting register so other nodes can execute the same method and include it in a new block if they receive a new optimum from an optimizer.

Blockchain Creation

Every time a new register in the *blocks* table is created, the creator node (the one that received the optimum), updates all the *accounting* table registers that have its field *block_belonging* as null to the *block_id* of the new block, these registers include logs of all requests made to nodes including evaluations requests and include the request that created the new optimum, after that a SHA-256 hash is calculated from the mentioned *accounting* registers concatenated with the hash of the previous block. For the current implementation of the platform, a hash of the entire current block is calculated, but a Merkle Tree as the one used in Bitcoin could be used If a large quantity of data per block is to be generated to save disk space as described in section 7 of the Bitcoin paper [44].

After this, the current block hash and the list of accounting register identifications (hashes) are updated in their respective fields of the new block register and the full block is flooded to other nodes using a JSON format thru the REST API, that after receiving the block will verify if they have all the accounting transactions in the list and if not, they request them from neighbors and execute them before creating the new block register and again flooding it. If some accounting registers with no block in a node are not included in a received block, the node must flood these registers, so other nodes can include them in the next block. The performance threshold is also a field of a block, this threshold can be calculated based on the previous block-time targeting a constant block time, but for the used validation application this is not required, and the threshold is the same performance of the optimum, so any new increments in performance will generate a new block.

Component-Level Architecture

Also, for authenticating, authorizing and registering the requests made by evaluators, tables for AAA are required on the node. The node uses an MVC architectural design pattern as shown in Figure 6. The clients, evaluators and optimizers send requests to the API endpoints, a request is composed of an HTTP method (such as POST), a route and the request parameters. The controller translates the endpoint routes to controller methods such as *GetParameter()*, and in the implementation of the mentioned controller methods, uses Active Record to create a data model and access an SQL database (SQLite was used), the use of Active Record makes the application database-engine portable. The controller uses the model of the required tables in the SQL database during its run-time. The responses of the database requests trough the models are used by the controller to generate the responses using parametrizable text templates with Nunjucks as views. The templates for the output are JSON-RPC v2.0 and SOAP v1.1 response messages that the optimizers, evaluators and clients receive from the nodes.

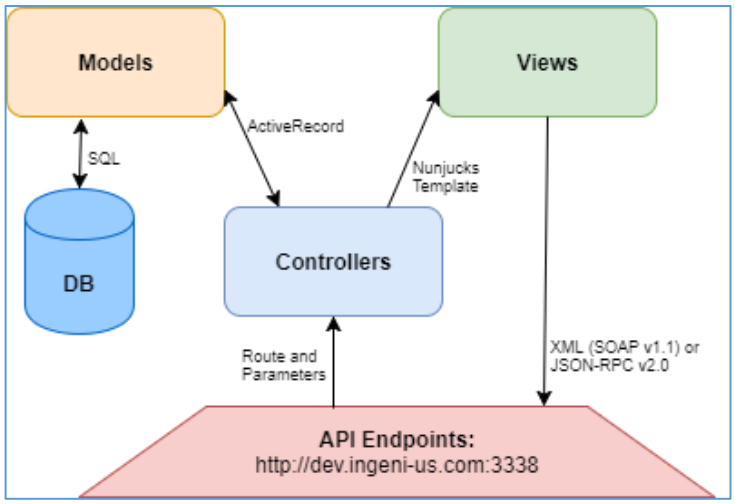

Figure 8- Component-level Architecture of Nodes

Web Interface

To allow the monitoring and administration of optimization processes, the nodes use the root route (/) to serve a Web interface with modules for viewing some variables such as the performance of the last optimum, and for creating, updating or deleting some values and configurations such as the table of neighbors that saves the IP addresses or URLs of known nodes, used for flooding. The Web interface also allows the creation of new users (allowed to make requests to the node) and their authorization as clients, optimizers, evaluators or nodes.

7.3.2. Optimizer

The optimizer reports to a node the best parameters it has found with better performance than the last ones already stored in the nodes. The optimizers can also retrieve the latest parameters from a node to try to optimize them further. In this section, the associated

requirements, architectural pattern and the selected technologies for implementation are shown.

7.3.2.1. Associated Requirements

Functional:

- The proposed software platform must allow the development of applications for optimization with distributed evolutionary computation using existing implementations of evolutionary algorithms.
- The way in which **optimizers** interact with the platform is by sending HTTP requests to a node for requesting the last optimization state or for posting a new optimum that has been found. They also report validation results (positive or negative) of the performance reported for some optimization state.
- The optimizers must authenticate when interacting with any of the nodes, so the nodes can perform authorization and log their interactions for traceability.
- The optimization is made by the optimizers by executing an evolutionary algorithm and synchronizing their optimization state between iterations of the EA.

Non-functional:

- Interoperability: The platform must provide a method for using heterogeneous implementations of an EA in multiple programming-languages.
- Traceability: All the operations between devices must be logged on to a ledger so historic data can be reviewed for debugging a dEC process and for accounting of services consumed and provided by participating devices.
- Scalability: New optimizers can be added to an existing optimization process providing additional computing capability to a dEC process.

7.3.2.2. Architectural Pattern

The architectural pattern that best fit the requirements of interaction via remote requests, authentication and interoperability is a Service Oriented Architecture (SoA) pattern because it allows a direct request of services to a node without intermediaries. The SoA pattern also fulfills the functional requirements by providing mechanisms of authentication, authorization and accounting. Also, a SoA pattern fulfills the non-functional requirements since it provides interoperability between multiple programming languages and the traceability can be implemented in a per-request basis.

7.3.2.3. Suitable Tactics

Orchestration is a tactic that uses a control mechanism (communications protocol) to coordinate, manage and sequence the invocation of services.

The required protocol is defined for satisfying the functional requirement of making evaluation requests or consulting results of an already requested evaluation. In the case of the optimizer, there are two protocols, one for retrieving last optimization state and other for reporting a new optimum.

7.3.2.4. Selected Technology

The available technologies for a SoA pattern implementation are Representational State Transfer (**REST**) [67] and Simple Object Access Control (SOAP) [68]. SOAP is an information exchange standard based on XML over HTTP, but it is more complex to implement and has more traffic overhead than **REST** due to the required XML plain text tags.

The selected technology is **REST** because it uses standard HTTP methods, it is relatively easy to implement in mainstream programming languages and it has descriptive methods (GET, POST, PUT, PATCH and DELETE) for the management of data collections.

The technology selected is REST that uses the standard HTTP methods, provides interoperability, allows authentication, authorization and accounting, and is simpler to implement than SOAP [68] while producing less traffic overhead [67].

7.3.2.5. Implementation

As the optimizers require to perform HTTP requests for both asking for the last optimization state and for posting a new optimum, the node must the same *parameters* table described in section 7.3.1.5.

The four methods identified on the node *GetProcess()* accessed by the route *GET /processes/id* where *id* is the process identifier (in the processes table in the node), this method returns the id of the most optimized parameters of a process from the parameters table in a node and *GetParameter()* accessed by the route *GET /parameters/id* that returns a register from the parameters table. The third method identified is *CreateParameter()* accessed by the route *POST /parameters* that allows to create a new parameter (and automatically a block if conditions are met). Finally, the fourth method is *InvalidResult()* that the optimizer must send if after the evaluation of some parameters, the reported fitness does not match the locally generated one, this method set the parameters as invalid in the node and they will not be included in the next block.

Also, for authenticating, authorizing and registering the requests made by evaluators, tables for Authentication, Authorization and Accounting (AAA) are required on the node.

7.3.3. Evaluator

The evaluator can request any node for a list of pending evaluations. From that list it selects one and performs the model evaluation with the data provided by the client and the latest parameters. Then it sends the results to any node, not necessarily the node that responded with the list, after that, the client that requested the evaluation can retrieve the results. In this section, the associated requirements, architectural pattern and selected technologies are shown:

7.3.3.1. Associated Requirements

Functional:

- The way in which an **evaluator** interacts with the platform is by sending remote requests to a **node** for retrieving a list of pending evaluations, retrieving data and model parameters for performing a pending evaluation or for posting the results of an evaluation.
- The client must authenticate when interacting with any of the nodes, so the nodes can perform authorization and log their interactions for traceability.

Non-functional:

- Interoperability: The platform must provide a method for using heterogeneous implementations of an EA in multiple programming-languages.
- Traceability: All the operations between devices must be logged on to a ledger so historic data can be reviewed for debugging a dEC process and for accounting of services consumed and provided by participating devices.

7.3.3.2. Architectural Pattern:

As in the optimizer, a Service Oriented Architectural (SoA) pattern is an adequate fit to fulfill the related requirements.

7.3.3.3. Suitable Tactics

As with the optimizer, orchestration is a suitable tactic to implement the desired behavior of the evaluator by defining protocols to coordinate, manage and sequence the invocation of services from a node.

The **protocol** is defined for satisfying the functional requirement of performing evaluation of pending requests, there are two procedures in an evaluator, the model parameters update and the pending evaluation execution, each one with its own **protocol** steps.

The parameters update protocol involves sending a request to a node for retrieving the last optimization state. The pending evaluation execution protocol involves retrieving a list of pending evaluations from a node, selecting the evaluation to perform using an evaluator-dependent criterion that can be the time the evaluation has been pending or by any order that the evaluator determines according to its convenience, evaluating the model with the provided parameters and inputs and posting the results to any node.

7.3.3.4. Selected Technology

As with the optimizer, the technology selected is REST that uses the standard HTTP methods, provides interoperability, allows authentication, authorization and accounting, and is simpler to implement than SOAP [68] while producing less traffic overhead [67].

7.3.3.5. Implementation

As the evaluators require to perform HTTP requests for both asking for pending evaluations to be performed and for posting the results of some evaluations, the node

uses the same *evaluation* table described in section 7.3.1.4 The evaluators also require consulting the most optimized parameters in an optimization process from any node.

The four methods identified on the node are *GetEvaluationList()* accessed by the route GET /evaluations/ and *UpdateEvaluation*() accessed by the route POST /evaluations/id, where *id* is the identifier of the evaluation obtained from the *GetEvaluationList()* method. The third method is *GetProcess()* accessed by the route GET /processes/id that returns the id of the most optimized parameter of a process with identifier *id* and GetParameter() accessed by the route GET/ parameters/id that returns a parameter with the identifier *id* int the table of parameters in all the nodes.

Also, for authenticating, authorizing and registering the requests made by evaluators, tables for AAA are required on the node.

7.3.4. Client

The client can request evaluations of optimized models with arbitrary data or view the status of an optimization process to select where he wants to request his evaluations. The client only has direct interaction with any of the nodes that orchestrate the pending evaluations to be fulfilled by the evaluators that use optimized model parameters produced by the optimizers. In this section, the associated requirements of the client, architectural pattern, suitable tactics for implementation and selected technologies are shown.

7.3.4.1. Associated Requirements

Functional:

- The client interacts with the platform by sending remote requests to any node for requesting an evaluation or consulting the results of an evaluation.
- The client must authenticate when interacting with any of the nodes, so the nodes can perform authorization and log their interactions for traceability.

Non-functional:

- Interoperability: The platform must provide a method for using heterogeneous implementations of an EA in multiple programming-languages. The client requires a method to interoperate with the system that is not programming-language dependent like using Web requests.
- Traceability: All the operations between devices must be logged in to a ledger so historic data can be reviewed for debugging a dEC process and for accounting of services consumed and provided by participating devices.

7.3.4.2. Architectural Pattern

As with the optimizer in section 7.3.2.2, the architectural pattern that is best fit for the requirements of interaction via remote requests, authentication and interoperability is a Service Oriented Architectural (SoA) pattern.

7.3.4.3. Suitable Tactics

As with the optimizer in section 7.3.2.3, orchestration: It is a tactic that uses a control mechanism (protocol) to coordinate, manage and sequence the invocation of services from other nodes and to respond to requests made by clients, evaluators and optimizers.

7.3.4.4. Selected Technology

The available technologies for a SoA pattern implementation are Representational State Transfer (**REST**) [67] and Simple Object Access Control (SOAP) [68]. SOAP is an information exchange standard based on XML over HTTP, but it is more complex to implement and has more traffic overhead than **REST** due to the required XML plain text tags.

The selected technology is **REST** because it uses standard HTTP methods, it is relatively easy to implement in mainstream programming languages and it has descriptive methods (GET, POST, PUT, PATCH and DELETE) for the management of data collections.

7.3.4.5. Implementation

The clients require to perform HTTP requests for 2 actions: asking for an evaluation to be performed and consulting the results of some previous evaluation (or verify if an evaluation has already been performed). Thus, the nodes must implement a table to register those requests and their results, this table is called the *evaluation* table.

The two methods identified on the node are *CreateEvaluation()* accessed by the route POST /evaluations. and *GetEvaluation()* accessed by the route GET /evaluations/id, where *id* is the identifier of the evaluation obtained from the *CreateEvaluation()* method.

Also, for authenticating, authorizing and registering the requests made by clients, tables for *Authentication*, *Authorization* and *Accounting* (AAA) are required on the nodes.

8. TECHNICAL DECISIONS

This section describes the technological decisions for the implementation and deployment of the proposed platform based on the technologies selected in the design process and the technological offer at the time of the writing of this document.

8.1. DEVELOPMENT AND EXPERIMENTAL ENVIRONMENT

For developing the code for the platform and for executing the empirical validation experiment, a development environment was used, it was composed of two computers hosting 3 virtual machines each one. The first host computer was a Lenovo® Y500 laptop computer featuring a quad-core Intel® Core™ I7 processor, 12GB of RAM and 750GB of HDD space with Microsoft® Windows™ 10. The second host computer was a custom-built desktop computer featuring a quad-core Intel® Core™ I7 processor, 6GB of RAM and 1TB of HDD space with Ubuntu Linux 17.10. These host machines were selected since they can execute up to 3 simultaneous Oracle® VirtualBox guest machines each one and they are property of the author. This same environment was used to perform the empirical validation experiment as described in section 9.

Oracle® VirtualBox version 5.2.0 was selected for executing a node and other programs to perform the validations experiments since it is open source software under the terms of the GNU General Public License (GPL) version 2 and this fact reduces implementation costs relative to commercial virtualization alternatives.

The 6 guest machines have 1 64bit core, 1.5GB of RAM, 20GB of storage capacity and bridged networking with connection to Internet. The aforementioned settings were selected for running Ubuntu Linux version 17.10 in the guest machines, this operating system was selected because it is open-source and the author have experience using it.

8.2. SOURCE CODE VERSION CONTROL

Git was used as source version control system and GitHub was used for hosting the source code because it is a widely used tool by developers and because of the author´s experience with this version control system. A GitHub Repository was created for hosting the code for the platform, tests and experiments.

8.3. PROGRAMMING LANGUAGES

The following sub sections describe the programming languages used for implementation and the justification of their selection.

8.3.1. Nodes

The JavaScript programming language was used for the implementation of the nodes on the Node.js version 8.11.1 cross-platform JavaScript run-time environment. Node.js was selected because it is open-source and it has an event-driven architecture capable of

asynchronous I/O, operating on a single thread with non-blocking I/O calls that allow thousands of connections without thread context switching that is mandatory in other environments for API development that use multiple threads.

The Adonis.js version 5.0.6 is a Model View Controller (MVC) framework for Node.js that allows code-reuse because its modular design and provides pre-tested components and Active Record to use in an application. The MVC architectural pattern divides the application into three components: model, to manage database connections, view, to manage output layouts; and controller, to orchestrate the communication between the former. Adonis.js was selected for the node's implementation because the use of a MVC architectural pattern allows ease of modification and reuse of components and the use of Active Record allows the use of seeds, migrations and factories that automate the initialization and population of testing databases from the source code without using SQL and making the application portable to multiple database engines. Also, the author has experience with the MVC architectural pattern.

SQLite 3 was used as database engine because it does not require additional services installed and because only local database access is required in each node.

8.3.2. Optimizers, Evaluators and Clients

Python 3.6.4 was selected for implementing the optimizers since it is open-source under the Python License, the author has experience with it and has support for the NEAT-Python v9.2 optimization library that will be used for evaluating and optimizing a neural network for automated forex trading.

9. VALIDATION EXPERIMENTS

The platform was implemented based on the proposed design, manual functional tests were performed during implementation to verify that the application had the expected behavior described by each functional requirement in section 7.1.4. After the implementation was finished, validation experiments were performed to verify that the non-functional requirements described in section 7.1.3 were met. In the following sections, these experiments are described, they were performed in the designed software platform using an optimization application. The section 9.1 describes why each experiment was made, what each experiment must do, what variables were measured and the expected results. The section 9.2 describes the experimental setup and deployment. Section 10 describes the results.

9.1. EXPERIMENTS DESCRIPTION

There are 3 experiments that were performed to empirically validate the following attributes in an application that uses the proposed proof-of-work, the requirement of Interoperability is not mentioned since it is implicitly satisfied by using a REST API:

1) **Scalability:** By adding nodes to the network, their combined computational capability should be incremented. (section 9.1.1)
2) **Fault-tolerance:** The removal of any of the nodes of the network must not stop the application from working. (section 9.1.2)
3) **Invalid result rejection:** The attempt to report an invalid result during the generation of an OPoW must be rejected by participating nodes. (section 9.1.3)

These experiments run on the same setup as development environment (section 8.1). In each of the virtual machines a node was configured using its Web interface, additionally each virtual machine executes an optimizer. Every node was configured to have 2 neighbors forming a ring topology to test the worst-case scenario for the controlled flooding of requests described in section 7.3.9.3 since they must be forwarded through many nodes and the network has a loop.

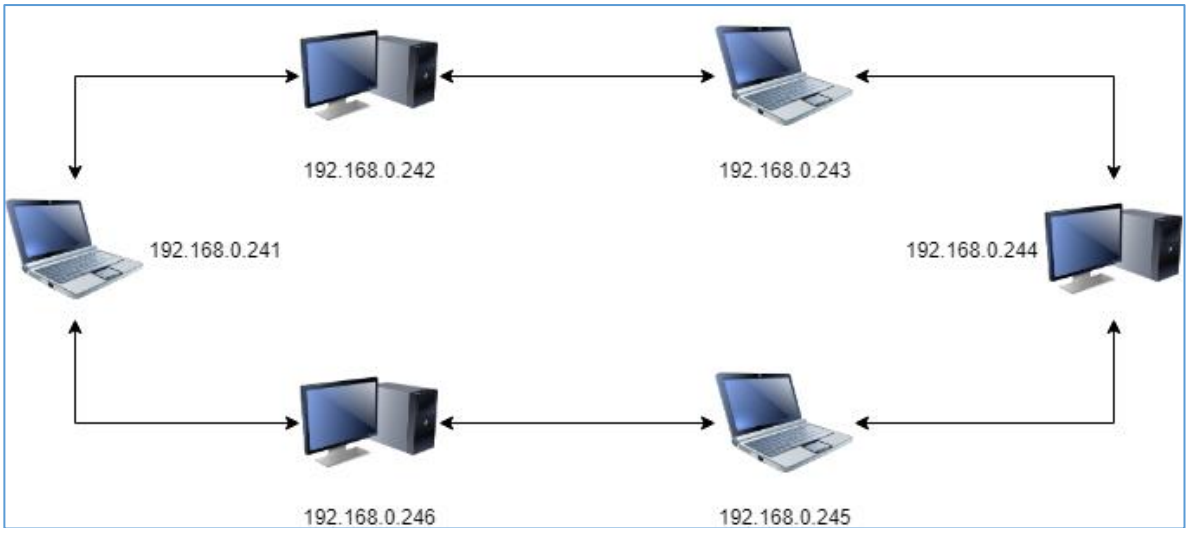

Figure 7 – Ring topology for the virtual machines used in the experiments

The application used in the experiments was the optimization of a neural network for controlling an automated trading agent in a forex account simulation using a neuroevolution technique called Neuroevolution of Augmenting Topologies (NEAT).

The forex account simulator used in the optimizers, loads a CSV with historical data, sets an initial capital in a simulated account and allows an external agent to buy or sell a lot of currency with a configured leverage. This simulator was implemented by the author as an environment called gym-forex for the OpenAI-Gym reinforcement learning framework.

Fitness Function

The fitness function (score) in the environment is empirical, it measures how well the trading is being made by the agent, is area under the curve of the equity plus the final balance. The optimization process tries to maximize the fitness.

$$fitness = \sum_{t=0}^{t=num\_ticks} \frac{\left(\frac{equity(t) - initial\_capital}{num\_ticks}\right) + (balance(t) - balance(t-1))}{2 \times initial\_capital}$$

Where *num_ticks* is the number of observations in the training dataset, *equity(t)* is the capital in the simulated account including the profits (or losses) of open orders at the time *t* in ticks, *balance* is the capital in the account without including open orders and *initial_capital* is the starting capital configured in the account when the forex account simulation starts.

The agent that controls the gym-forex environment is also used in the optimizers for generating a trained neural network that takes the best action possible for an observation feed to its inputs. The agent is a modification of the one made for the Lunar Lander v2 environment that comes with the NEAT-Python library v0.92.

The modifications made to the original Lander agent were the adaptation from the Lunar Lander to the Forex environment in the observation space, that was composed by the game variables and had to be modified to include the order status, also the action space that before was composed of the controls of the Lander ship and now is a control for buying or selling and the addition of a *migration* function at the start of each iteration of the NEAT algorithm used in Lander to synchronize the best genomes between each optimizer and the rest of the network.

In this setup, the three experiments were performed, the procedures to perform each experiment and the variables that were measured are described in the following subsections. All the experiments used data from HistData.com for the EUR/USD currency pair between January 1 of 2017 until December 31 of 2017 with 1-minute resolution. The data contained fields for timestamp, the bid prices for high, low, open, close and volume of transactions for each 1-minute period. The data was imported to a MQL program made by the author to down sample the data to 1 hour resolution and to calculate technical indicators to be used as features of the training dataset. The test data had 19 features in total generated by the MQL program including: the high, low, close, volume of transactions, hour of the day, day of the week, day of the month, and the technical indicators parametrized for the short term (12h) and long term(120h): Exponential Moving Average, Relative Strength Index, Moving Average Convergence, Average Directional Index, Commodity Channel Index and the Average True Range Index.

Each experiment has multiple stages. After all the programs (optimizers, nodes and evaluators) start working with a reset script, the change from one stage to another in the experiments are made manually via configurations of each node with the Web interface of the platform. As the experiments require to repeat the optimization process many times, a configurable script was made in Python to and save a copy of the database before repeating the optimization process a defined number of times.

9.1.1. Scalability Experiment

In this experiment, the scalability of the optimization process is validated by measuring the reduction in training time by incorporating new optimizers to the same optimization process. All the measurements are made by using the Web interface of the proposed platform.

Stage 1: Six optimizers were setup to work for 3 hours on the gym-forex NEAT agent and after that period the performance is measured. The task is repeated 10 times to obtain an average of the performance reached during a 3-hour period.
Stage 2: One isolated optimizer was setup and 10 executions of 3-hour periods of training were made and after that period the performance is measured. The task is repeated 10 times to obtain an average of the performance reached during a 3-hour period.

The expected result of this experiment is that the performance in the 6 optimizers cooperatively should be larger than the one reached by the single optimizer test, and therefore the training time to reach the performance obtained by a single optimizer was reduced if more optimizers are incorporated.

9.1.2. Fault-Tolerance Experiment

An existing optimization process must continue to execute in the event of failure of any of the nodes participating in it. The performance measurements are made by using the Web interface of the proposed platform.

Stage 1: Six optimizers were setup to work for 30 minutes on the gym-forex NEAT agent as show in Figure 7, at the end of this period, the performance is measured and 3 of them are isolated from the other 3, 2 of the isolated ones keep communication between them and the other one was isolated with its optimization process stopped as shown in Figure 8.

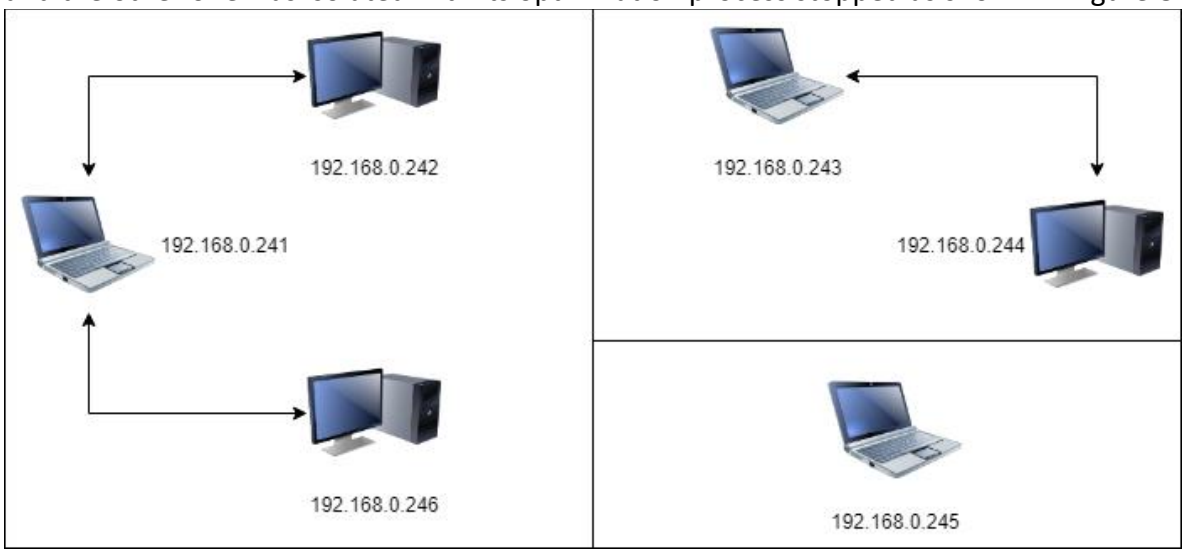

Figure 9 - Network configuration for the fault-tolerance experiment

Stage 2: The new network configuration is left to work for another 30 minutes after the end of stage 1, at the end of this period, the performance in each optimizer is measured again and the connections are reestablished as in the start of stage 1

Stage 3: The new network configuration is left to work for 30 minutes, and after that period a last measurement of the performance is made.

The 3 stages are repeated 10 times to obtain an average of the performance reached during each of the stages of the test.

The expected result of this experiment is that the population of optimization states (populations of mathematical model parameters such as neural network weights) synchronize their best performer specimens between optimizers by incorporating to the population composing the optimization of the nodes, the last optimum reported in the

blockchain after reconnection and the best performance between the isolated parts is used in all the optimizers in the network.

9.1.3. Invalid Result Rejection Experiment

An existing optimization process must ignore invalid optimization states shared by optimizers. The performance measurements are made by using the Web interface of the proposed platform.

Stage 1: Six optimizers were setup to work for 1 hour on the gym-forex NEAT agent, at this moment the performance is measured and at the end of this period, a specially crafted API request with incremented performance than the real one is sent from an optimizer.
Stage 2: The network is left to work for a 30-minute period, the performance is measured again one last time at the end of this period.

The stages 1 and 2 are repeated 10 times to obtain an average of the performance reached during each of the stages of the test.

Stage 3: As control measurements, the experiment in the six optimizers is restarted and they are left to work for 1 hour and 30 minutes, the performance is measured at the end of this stage.

The stage 3 is repeated 10 times to obtain an average of the performance reached during this stage. The request that introduced the invalid request was logged but its result was not used by the optimizers since they continued optimizing from the last valid optimization state (from the last block), ignoring the invalid result.

The expected result of this experiment is that the result in stage 3 must be similar to the one at the end of the stage 2, since the invalid result does not affect the optimization process since the optimizers verify the reported parameters by evaluating them before incorporating them to their optimization state.

9.2. EXPERIMENTAL SETUP

Ubuntu 17.10 was installed in one of the virtual machines, in this machine a node was installed, also in the machine an optimizer was installed with both the forex-gym environment and the NEAT-agent cloned from their GitHub repositories. The required system packages were installed using *aptitude*, the node.js packages were installed using *npm* and the python packages were installed using *pip3*.

After installing one virtual machine, 5 clones were made and in each one a different static IP address was configured.

After installation, reset scripts for automatically pulling the last version from GitHub, re-populate the database and start the program are created for the node and the optimizer

(NEAT-agent). To start each experiment, the reset script was executed manually in all the virtual machines.

10. RESULTS AND ANALYSIS

This section contains the measurements obtained during the experiments described in section 9 and an analysis of them that will be expanded in the discussion on section 11. The performance is an average between the final equity and the average equity during each environment simulation as defined in the fitness function in section 9.1.

10.1. SCALABILITY EXPERIMENT

Table 1 shows the average performance in each stage of the scalability experiment.

Stage	Number of Optimizers	Avg. Performance
1	6	3.28
2	1	0.76

Table 1 - Scalability Experiment Results

The decreased average performance of the stage 2 of this experiment empirically validate that the 6 devices used in stage 1 are cooperating and their combined computing capacity made a larger advancement in optimization in the same amount of time.

10.2. FAULT-TOLERANCE EXPERIMENT

Table 2 shows the average performance in each stage of the fault-tolerance experiment for each group of isolated optimizers.

Stage	Time [minutes]	3 Optimizers Group	2 Optimizers Group	1 Optimizer
1	30	2.48	2.48	2.48
2	60	2.61	2.53	2.48
3	90	2.96	2.96	2.96

Table 2 – Fault-tolerance Experiment Results

At the end of stage 2, the set of two isolated optimizers reaches a different performance compared to the set of three optimizers, but during stage 3 the optimization states are synchronized, and all the previously isolated device sets have the same performance in their optimization processes. The conclusion is that the optimization process continues as expected after a communications fault occurred.

10.3. INVALID RESULT REJECTION EXPERIMENT

The table 3 shows the average performance in each stage of the experiment and in the control group.

Stage	Avg. Performance	Control
1	2.95	2.91
2	3.11	3.24

Table 3 – Invalid Result Rejection Experiment Results

The performances reached in stage 2 and stage 3 are similar, indicating that the optimizing process was able to ignore the invalid result introduced at the end of stage 1 and continue to execute normally.

11. DISCUSSION

The proposed platform provides a REST API that is executed on a network of nodes connected to each other through a network with a decentralized architectural pattern and allows the developers to implement evolutionary algorithms, adding scalability, fault-tolerance and traceability that a decentralized architectural pattern provides. The REST API is a programing-language agnostic method to synchronize the optimization state with other optimizers in the network by migrating the best performing specimens from an optimizer to another between iterations of an evolutionary algorithm.

Since all the evolutionary algorithms are based on having a population of candidate solutions that evolve with iterations of some steps (selection, crossover, mutation), the synchronization of optimization states can be made between iterations for making the population have always the best performing specimens in the network and avoiding other optimizers to having to find the optimums for themselves and becoming a way to provide scalability for an EC process as shown by the experiments in section 9.

The results of the experiments were as expected, since the island model allows migrations for providing scalability to an optimization process while a decentralized architectural pattern provides fault tolerance, and the use of an optimization PoW provides the block-time control required to use a blockchain for traceability of application events and optimization states in a decentralized architecture. The optimization states stored in the blockchain can be verified independently by other optimizers to allow the rejection of invalid results.

The platform, apart from allowing decentralized optimization for scalability and fault-tolerance, allows the use of optimized model parameters from the evaluators, in this case to predict the best action to take in any moment given some market and current order status information. The use of evaluators provides additional functionality to the developed platform since external applications can use the most optimized model at any time no matter if the optimization process is still working.

As shown in the experiments, the benefits of a decentralized architectural pattern can be achieved for a distributed optimization process with EC, using a blockchain to provide traceability of information in this case, using an optimization-based proof of work to control the block-time without using a cryptographic proof of work.

The use of the proposed platform can allow the collaboration between peers in optimization applications or experiments, with scalability, fault-tolerance, invalid result rejection, a REST API for easing the integration of the platform with existing evolutionary algorithms from most programming languages and a Web based interface that uses the API for monitoring and configuration of optimization processes.

In exchange of the security of the CPoW by the usability of the OPoW for optimization, it is uncertain and up to be researched what level of security is acceptable for a cryptocurrency if the security is traded for usability, since the value of the cryptocurrencies, as with the value of everything else in a market, depends on the demand it has, and the demand of a cryptocurrency usually is dependent on how secure it is, since the users do not want to lose their money, but still the OPoW is an alternative for escalating dEC optimization processes, making them fault-tolerant and traceable, an attribute that can be useful to debug, test and deploy applications based on optimization with dEC.

12. FUTURE WORK

Updates to the platform can be future work, for increased security and for exploring the use of hybrid hash-based and optimization-based proof-of-work generation, having both a performance increment and the leading number of zeroes in a hash as difficulty for controlling the tradeoff between security and usefulness of a proof-of-work in a decentralized optimization application as mentioned in section 6.4.

The tradeoff between security and usability hypothetically could be controlled by having the optimizer to produce both an OPoW by generating an optimization state with a performance superior to the block-creation threshold and a CPoW similar to the used in Bitcoin with its independent difficulty consisting of the number of leading zeroes of a hash of a block containing both the previous block hash and the OPoW that is a hash of the block contents that must contain the last optimum report request from an optimizer.

This approach will allow to have the security provided by a CPoW, at the expense of a reduced difficulty, in exchange of the usability of the OPoW for optimization, it is uncertain and up to be researched what level of security is acceptable for a cryptocurrency if the security is traded for usability.

Future work also includes the comparison between different migration functions in the performance of an optimization process since migrating a different number of specimens may or may not impact the time to increment the performance of some parameters of a mathematical model being optimized. Also, future work includes to find the best number of iterations to skip between migrations and to test different evolutionary algorithms and multi-level optimization techniques that use multiple parallel optimization processes such as CoDeepNEAT [71].

13. CONCLUSIONS

The main conclusion of this work is that it is possible to use an optimization-based proof of work to control the block-time of a blockchain that is used to synchronize the last optimization state between optimizers and to trace the changes in an optimization state during an optimization process, the optimization state can be verified independently by other nodes and because of this, the participating nodes do not need to trust each other.

An advantage of the platform developed to empirically validate the proposed proof-of-work is that it allows developers to make scalable, fault-tolerant and traceable implementations of evolutionary algorithms. Its main disadvantage is the security tradeoff that must be made for producing a useful proof-of-work that is not function of the contents of the block, and additional security measurements could be required depending on the application.

Even if the OPoW cannot be used to substitute CPoW for cryptocurrency mining because of the OPoW reduced security relative to CPoW, it still can be of use in cases of optimization applications that require scalability and fault tolerance, in which the security constraints of the OPoW are non-priority or the application allows some way to mitigate them.

The design of a platform for empirically validating the proposed proof-of-work, was used to confirm the expected results of the validation experiments, but also adds the capability of evaluating an external result in a model that was optimized or is being optimized in a decentralized network, while saving the register of this evaluation in the blockchain. This can be useful for the development and debugging of multi-level optimization applications.

14. REFERENCES

[1] K. Stanley, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation,* vol. 10 , no. 2, pp. 99-127 , 2002.

[2] L. Wu, "Magnetic Resonance Brain Image Classification by an Improved Artificial Bee Colony Algorithm," *Progress in Electromagnetics Research,* vol. 116, pp. 65-79 , 2011.

[3] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Neural Networks*, 1995.

[4] M. Wa, "Genetic Algorithm and its application to Big Data Analysis," *International Journal of Scientific & Engineering Research,,* vol. 5, no. 1, 2014.

[5] P. Verbancsics, "Classifying Maritime Vessels from Satellite Imagery with HyperNEAT," in *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference - GECCO Companion*, New York, USA, 2015.

[6] Y.-J. Gong, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art.," *Applied Soft Computing,* 2015.

[7] J. Lienig, "A parallel genetic algorithm for performance-driven VLSI routing," *IEEE Trans. Evol. Comput,* vol. 1, no. 1, pp. 29-39, 1997.

[8] H. Piereval, "Distributed evolutionary algorithms for simulation optimization," *IEEE Trans Sys. Man. Cybern. ,* vol. 30, no. 1, pp. 15-24, 2000.

[9] K.C.Tan, "Automating the drug-sheduling of cancer chemotherapy via evolutionary computation," *Artif.Intellig.Med.,* vol. 25, no. 2, pp. 169-185, 2002.

[10] J. Creput, "Automatic mesh generation for mobile network dimensioning using evolutionary approach," *Evol.Comput,* vol. 9, no. 1, pp. 18-30, 2005.

[11] J.Liu, "An evolutionary autonomous agents approach to image feature extraction.," *IEEE Trans. Evol. Comput.,* vol. 1, no. 2, pp. 141-158, 1997.

[12] M. Epitropakis, "Hardware-friendly higher-order neural network training using distributed evolutionary algorithms.," *Appl. Soft. comput. ,* vol. 10, no. 2, pp. 398-408, 2010.

[13] L. Kattan, "Distributed evolutionary estimation of of dynamic traffic origin/destination," in *13th IEEE Conference on Intelligent Transport Systems*, 2010.

[14] J. Noyima, "Ensemble classifier design by parallel implementation of genetic fuzzy rule selection for large datasets.," in *IEEE Congress on Evolutionary Computation*, 2010.

[15] F. Rainville, "DEAP: A Python Framework for Evolutionary Algorithms," in *GECCO*, 2012.

[16] M. Linder, "Grid computing in Matlab for solving evolutionary algorithms," in *Technical Computing Bratislava*, Bratislava, 2012.

[17] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Neural Networks.* , 1995.

[18] D. D. Karaboga, "An Idea Based On Honey Bee Swarm for Numerical Optimization," Erciyes University, 2005.

[19] M. Dorigo, "Distributed Optimization by Ant Colonies," in *Première conférence européenne sur la vie artificielle*, Paris, France, 1991.

[20] C. Huang, "A hybrid stock selection model using genetic algorithms and support vector regression," *Applied Soft Computing,* vol. 12, p. 807–81, 2012.

[21] E. Levy, "Genetic algorithms and deep learning for automatic painter classification," in *conference on Genetic and evolutionary computation - GECCO '14* , New York, New York, USA, 2014.

[22] E. Levy, "Genetic algorithms and deep learning for automatic painter classification," in *Proceedings of the 2014 conference on Genetic and evolutionary computation - GECCO '14* , New York, New York, USA, 2014.

[23] S. Whiteson, "Evolutionary Function Approximation for Reinforcement Learning," *Journal of Machine Learning Research,* vol. 8, pp. 877-917, 2006.

[24] R. B. Greve, "Evolving Neural Turing Machines for Reward-based Learning," in *Genetic and Evolutionary Computation Conference (GECCO)*, Denver, Colorado, USA, 2016.

[25] A. Santoro, "One-shot Learning with Memory-Augmented Neural Networks," Cornell University , Ithaca, NY, USA, 2016.

[26] D. Izzo, "The generalized isalnd model," *Studies in Computational Intelligence,* vol. 415, pp. 151-169, 2012.

[27] G. Folino, "P-CAGE: An Environment for Evolutionary Computation in Peer-to-Peer Systems," in *European Conference on Genetic Programming*, Berlin, 2006.

[28] A. L. Ian Scriven, "Decentralised distributed multiple objective particle swarm optimisation using peer to peer networks," in *IEEE World Congress on Evolutionary Computation, 2008. CEC 2008*, 2008.

[29] A. L. C. Fernando Silva, "odNEAT: An Algorithm for Decentralised Online Evolution of Robotic Controllers," *Evolutionary Computation ,* vol. 23, no. 3, pp. 421-449, 2015.

[30] D. Jakobović, "ECF - Evolutionary Computation Framework," University of Zagreb, http://ecf.zemris.fer.hr/, 2014.

[31] M. A. García-Sánchez P., "A Methodology to Develop Service Oriented Evolutionary Algorithms," *Intelligent Distributed Computing VIII,* vol. 570, 2015.

[32] S. Cahon, "ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics," *Journal of Heuristics,* vol. 10, no. 3, pp. 357-380, 2004.

[33] D. R. White, "Software review: the ECJ toolkit," *Genetic Programming and Evolvable Machines,* vol. 13, no. 1, pp. 65-67, 2011.

[34] M. G. Arenas, "A Framework for Distributed Evolutionary Algorithms," *Lecture Notes on Computer Science,* vol. 2439, pp. 665-675, 2002.

[35] J. L. J. Laredo, "Resilience to churn of a peer-to-peer evolutionary algorithm," *International Journal of High Performance Systems Architecture,* vol. 1, no. 4, pp. 260-268, 2008.

[36] C. Rohrs, "Query Routing for the Gnutella Network," Lime Wire LLC, 2002.

[37] R. Fielding, "Chapter 5: Representational State Transfer (REST)," in *Architectural Styles and the Design of Network-based Software Architectures*, Irvine, Ca, University of California, 2000.

[38] P. C. R. K. Len Bass, Software Architecture in Practice - Second Edition, Addison Wesley, 2003.

[39] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash Syste," *bitcoin.org,* vol. Retrieved 28 April 2016.

[40] H. C. M. Kalodner, "An empirical study of Namecoin and lessons for decentralized namespace design," in *Workshop on the oconomics of information security*, Delft, The Netherlands, 2015.

[41] D. Carboni, "Feedback based Reputation on top of the Bitcoin Blockchain," Retrieved from http://arxiv.org/abs/1502.01504 .

[42] E. Foundation, "Ethereum's white paper," Ethereum Foundation, 2014.

[43] G. N. O. Zyskind, "Decentralizing Privacy: Using Blockchain to Protect Personal Data," *IEEE Security and Privacy Workshops ,* p. 180–184, 2015.

[44] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," in *Web. bitcoin.org. Retrieved 28 April 2014*, 2008.

[45] S. King, "Primecoin: Cryptocurrency with Prime Number Proof-of-Work," 2013 .

[46] A. R. Marshall Ball, "Proofs of Useful Work," *Cryptology ePrint Archive,* 2017.

[47] R. A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, Massachusetts: The MIT Press, 2005.

[48] M. S. J. Moody, "Learning to trade via direct reinforcement," *IEEE Transactions on Neural Networks,* vol. 12, no. 4,, 2001 .

[49] C. WATKINS, "Q,-Learning," *Machine Learning,* vol. 8, 1992.

[50] R. M. Kenneth O. Stanley, "Efficient Reinforcement Learning through Evolving Neural Network Topologies," in *Genetic and Evolutionary Computation Conference* , San Francisco, CA, 2002.

[51] C. Igel, "Neuroevolution for reinforcement learning using evolution strategies," in *Congress on Evolutionary Computation*, 2003.

[52] V. Heidrich-Meisner, "Neuroevolution strategies for episodic reinforcement learning," *Journal of Algorithms,* vol. 64, no. 4, pp. 152-168, 2009.

[53] D. Lessin, "Open-ended behavioral complexity for evolved virtual creatures," in *15th annual conference on Genetic and evolutionary computation*, Amsterdam, The Netherlands, 2013.

[54] G. B. a. V. Cheung, "OpenAI Gym," *Arxiv,* vol. arXiv:1606.01540, 2016.

[55] N. G. L. Iker Zamora, "Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo," *Whitepaper,* 2016.

[56] A. H. David Silver, "Mastering the game of Go with deep neural networks and tree

search," *Nature,* vol. 529, no. 7587, pp. 484-489, 2016.

[57] K. K. Volodymyr Mnih, "Human-level control through deep reinforcement learning," *Nature,* vol. 518, no. 7540, pp. 529-533, 2015.

[58] U. B. o. E. Analysis, "U.S. INTERNATIONAL TRADE IN GOODS AND SERVICES - December 2013," U.S. Department of Commerce, Washington, DC 20230, 2013.

[59] E. Department, "Triennial Central Bank Survey of foreign exchange and OTC derivatives markets in 2016," Bank For International Settlements, Basel, Switzerland, http://www.bis.org/publ/rpfx16fx.pdf - Retrieved on 20 March of 2017.

[60] Investopedia, "Forex Broker Definition," http://www.investopedia.com/terms/forex/c/currency-trading-forex-brokers.asp, Retrieved on March 20 of 2017.

[61] Investopedia, "Margin Call Definition," http://www.investopedia.com/terms/m/margincall.asp, Retrieved on March 3 of 2017.

[62] Investopedia, "What is a Spread," http://www.investopedia.com/terms/s/spread.asp, Retrieved on March 3 of 2017.

[63] Investopedia, "Leverage Definition," http://www.investopedia.com/terms/s/spread.asp, Retrieved on March 20 of 2017.

[64] A. P. CHABOUD, "Rise of the Machines: Algorithmic Trading in the Foreign Exchange Market," *Journal of the American Finance Association,* vol. 69, no. 5, p. 2045–2084 , 201.

[65] M. A. H. Dempster, "An automated FX trading system using adaptive reinforcement learning," *Expert Systems with Applications,* vol. 30, no. 3, pp. 543-552, 2006.

[66] HistData.com, "Free Forex Historical Data Repository," http://www.histdata.com/, Retrieved on Mach 20 of 2017.

[67] R. T. Fielding, "Chapter 5: Representational State Transfer (REST)," in *Architectural Styles and the Design of Network-based Software Architectures*, Irvine, University of California, 2000.

[68] H. W. Group, " SOAP: Simple Object Access Protocol," [Online]. Available: https://tools.ietf.org/html/draft-box-http-soap-00.

[69] e. a. Don Box, "Simple Object Access Protocol (SOAP) 1.1," W3C, 8 May 2000. [Online]. Available: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/. [Accessed 18 04 2018].

[70] J.-R. W. Group, "JSON-RPC 2.0 Specification," JSON-RPC Working Group, 04 01 2013. [Online]. Available: http://www.jsonrpc.org/specification. [Accessed 18 04 2018].

[71] R. Mikkulainen and J. e. a. Liang, "Evolving Deep Neural Netwokrs," *eprint arXiv:1703.00548,* 2017.