Bilkent University

Department of Computer Engineering

Senior Design Project

CrypDist

Project Design Report

Gizem Çaylak
Oğuz Demir
Turan Kaan Elgin
Mehmet Furkan Şahin
Onur Uygur

Supervisor: Can Alkan
Jury Members: İbrahim Körpeoğlu and Ercüment Çiçek

Analysis Report December 30, 2016

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2

1. Introduction

Data management has a great importance for most of the companies to maintain their business. Since technology is developing and companies are growing every day, it creates a demand to manage a huge amount of data. Managing huge amount of data is still an on-going research field and engineers are working on this field to provide secure and reliable systems with faster response time for their customers. One of the methods to store the data is data distribution which distributes the data on different servers to manage the data. Data encryption and data replication can be used together with data distribution to ensure the secure management of distributed data and the reliability of the system.

Our project, CrypDist, aims for maximizing the performance of the delivery of data and providing fast access for the users by using a decentralized distribution system.

1.1. Purpose of the System

The system is designed to implement a solution for providing ease of access to the genotype and phenotype data for researchers to enable them to conduct advanced research by analyzing and updating the data. This will be achieved by delivering a software system which will interact with the users to perform the operations on data. The system has a distributed database called "blockchain" to ensure the safety and reliability of continuously growing data.

1.2. Design Goals

1.2.1. Performance Criteria

- **Response time:** Time for responding to user must be as fast as possible such that a purpose of the data distribution is to provide data faster by using the nearest server as possible. In general, the response time should be less than 5 minutes assuming the data is large.

- **Memory:** For the system, 40 GB disk space, 4 GB RAM, and 2 CPU cores are needed.

1.2.2. Dependability Criteria

- **Security:** For reaching to authenticated data, clients should use their account information. Passwords of the clients should be stored in a secure database. In addition to that, authenticated data should be encrypted within the system for avoiding other clients to reach.

- **Fault Tolerance:** In case of server failures, system should replicate the data into backup servers.

- **Availability:** The system should be available all the time by the use of data replication and recovery of the data when a server failure is detected.

1.2.3. Maintenance Criteria

- **Portability:** It is an important goal such that the system will have more users if it is portable among platforms. Since Java Virtual Machine is platform independent, portability of the system can be managed.

- **Modifiability:** In the architectural pattern, subsystems should be separated from each other and layering should be used accordingly.

- **Adaptability:** The data distributed by the system should be a generic data which means the system should not know its application domain. By that way, the system can be used in many areas.

1.2.4. End User Criteria

- **Utility:** The system should help people to access data in a fast and secure way. It can be used for massive data related research areas like genomics.

- **Usability:** The system should have a user-friendly interface such that users should be able to generate their requests easily. However it is required for them to know SQL.

1.3. Definitions, Acronyms, Abbreviations

HTTP: Hypertext Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems.

HTTPS: Secure Hypertext Transfer Protocol is a protocol for secure communication over a computer network which is widely used on the Internet

1.4.Overview

CrypDist is a blockchain based distribution application that will be created with Java on the both client and server side. It is also going to use Akamai to handle the distribution part as an off the shelf component. CrypDist is going to use NetStorage facility of Akamai to store the raw data and it is going to distribute it with the help of Akamai Edge servers.

On the client side, CrypDist will be able to provide a summary data in form of a blockchain so that the user will be able to understand the content of the raw data itself without looking at the exact data. Blockchain structure is a data structure used for storing continuously growing list of records without tampering or revision. Hence, by the help of blockchain structure, CrypDist will be able to maintain the data dynamically and the synchronization will be handled automatically. Clients will be able download whatever part of the data they need, manipulate it and upload it back to servers to be reached in future as a new version. However, the data in the blockchain cannot be manipulated by anyone. In case of data addition, a block containing the abstract information will be added synchronously to blockchain on clients and the actual data will be distributed among servers. Thus, the encryption of the data will be already done by blockchain meaning that the user will never know where the data sent but he is always going to have the information in his own summary, blockchain. However, the raw data will already be open to anyone who has the blockchain. Here, CrypDist classifies the raw data to two different classes; one will include the genomics data which will be open and the second will include phenotype of the owner of the data which will be kept encrypted because it contains sensitive personal data. Thus, when a client wants to reach phenotype data, he will need to be authenticated by our services.

The main functionality of CrypDist is that it avoids the centralized database architecture so that we can have a democratized data distribution system which avoids any block and always up to deliver the data as possible as quickly thanks to Akamai services. Our first and main focus is to manage genomics data since demand to work on that specific data is huge such that many researchers are working to avoid future possible mutations by examining the outputs of the past research.

The problem which CrypDist is planning to solve is currently tried to be handled with dbGaP system, which provides centralized database of Genotypes and Phenotypes. Using dbGaP has several disadvantages: Firstly, using one centralized server for taking data will be slow, when number of clients and size of the data is considered. Secondly, dbGaP stores all the data secure and getting access to this data requires several diplomatic steps, which takes very long time. However, the accessibility of Genotypes data does not need to be controlled because it does not contain any personal information and it is needed for statistical data. As a result, currently researchers are having problem for accessing the secure Genotypes data, which should have been public. Lastly, since the data is stored in centralized database, crashes, in-tolerated faults or political issues (e.g. shut down applications by governments) on the servers may cause loss of data, which the Wellcome Trust users had encountered in the U.K.

Previously, blockchain structure was used in genomics for distributing CPU load between machines by exchanging bitcoins with process power (http://dnadigest.org/a-new-multi-centralized-cryptocurrency-coinami/). However, CrypDist is the first application using blockchain for data distribution.

2. Current Software Architecture

The blockchain is the key element of the digital currency system bitcoin. The blockchain is a distributed database. Bitcoin has a decentralized system because blockchain stores the data across its nodes of network. Bitcoin uses blockchain to record the bitcoin transactions between users. Network of communicating nodes provide maintenance (replication of the transactions and sending the transaction information to other nodes) for the blockchain. When a new transaction is added to a node, this transaction is broadcasted to other nodes. Each node in the network stores its own copy of blockchain. Although bitcoin uses blockchain to store the information about transactions, there is no current system keeping abstract data of human genome in its blockchain.

3. Proposed software architecture

3.1. Overview

CrypDist is a system which provides mass distributed data storage and synchronized data storage among clients where clients can interact with synchronized local data and mass data via graphical user interface. For this reason, the system is divided into subsystems according to functionality, in order to split the developing complexity of the project. Furthermore, with this decomposition, the independency of functional parts is being tried to be increased so that a subsystem can be changed without changing other subsystems, which increases the extendibility and maintainability of the system.

The main functionalities that CrypDist aims to satisfy are Graphical User Interface, Synchronized Data Storage among clients (BlockChain), local data storage for storing BlockChain data and server side storage where the mass data will be stored.

3.2.Subsystem decomposition

The CrypDist system is decomposed into subsystems according to functionalities. For client software which consist of GUI, BlockChain and Local Data Storage subsystems, 3-Tier Architecture will be followed in order to increase the extendibility and maintainability of the system, as mentioned above. The server software will work independently from client side.

3.2.1.Graphical User Interface Subsystem

Graphical User Interface Subsystem is “Presentation” layer of 3-Tier Architecture and forms the outside door of the system where the users can interact with the system. The information about BlockChain is visualized in interface and users can perform actions (query the data, download data, update data, store new one) through this interface.

3.2.2.BlockChain Subsystem

BlockChain Subsystem is “Application” layer of 3-Tier Architecture and it is responsible for synchronizing the summary data among the clients. In initialization step of system, this subsystem read the locally stored data from Local Data Storage Subsystem, in order not to re-download the whole data. For synchronizing the data, this subsystem establishes peer to peer communication between clients. Also, for downloading and updating the distributed mass data, this subsystem has a connection to Mass Data Storage Subsystem. For this connection, Client-Server architecture will be applied and BlockChain Subsystem will form the client side of this connection.

3.2.3.Local Data Storage Subsystem

Local Data Storage Subsystem is “Data” layer of 3-Tier Architecture and it stores stores the BlockChain data in local databases which prevents system from re-downloading whole data after first initialization. which prevents system from re-downloading whole data after first initialization. The local data is updated whenever there is an update on BlockChain, and it is kept synchronized with BlockChain as system is alive.

3.2.4.Mass Data Storage Subsystem

Mass Data Storage Subsystem is server side of the system and responsible for storing the whole data which cannot be stored locally on clients. This subsystem forms server part of the Client – Server architecture (between BlockChain and Mass Data Storage) which is stated in BlockChain subsystem. This subsystem works independently from client software and it provides download, update and upload links for client software to make the clients able to manipulate the mass data.

3.3.Hardware/Software Mapping

Diagram 1 summarizes the general deployment structure of CrypDist including Distribution system, Client application and the blockchain structure.

CrypDist consists of two main components as following;

- Distribution System
  - A de-centralized distribution structure is going to be serving clients. (The component is powered by Akamai Technologies)
  - Database
- CrypDist Application (Client Side)
  - Blockchain
  - CrypDist management panel

Client's computer is going to be connected to the other clients with HTTP protocol by the peer to peer design for being updated by the ongoing updates made by the other clients on the raw data. The client application's internet connection module is going to be always listening for requests from other clients. Raw data is going to be stored in distribution system which includes servers located in different parts of the world. Each client is going to be connected to the distribution system with HTTP protocol and requests to download/upload data are going to be made by this channel. In servers, one database application and one connection module are going to be running and listening for requests. The server application is going to fetch/store the data from/in the database according to the requests made by the connection module.

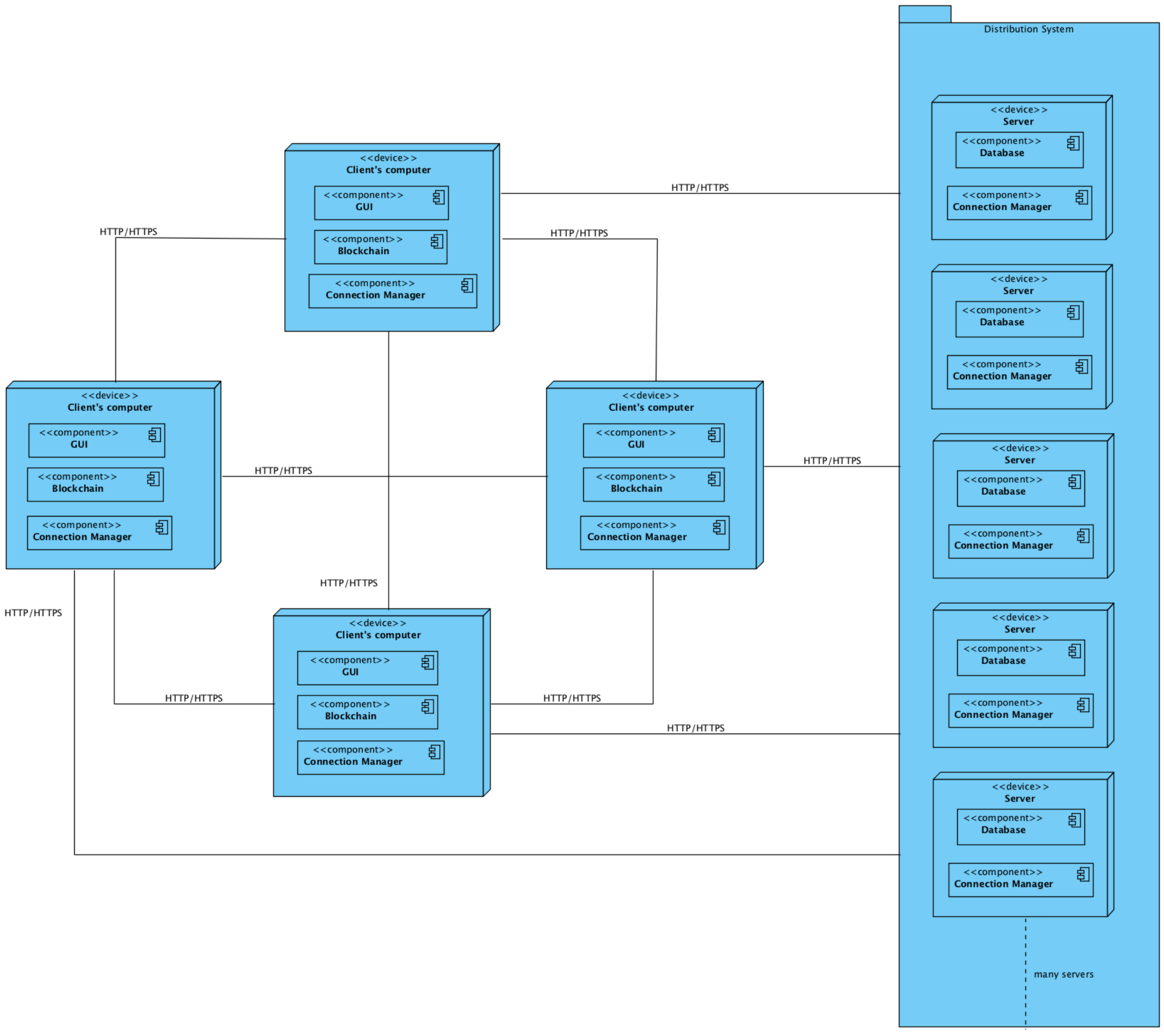

Diagram 1: Hardware Mapping

3.4.Persistent Data Management

Raw data storage will be provided by Akamai. In the system, data summaries will be kept. Since there are many clients, the system should support concurrent accesses for data. There will be some queries on the data attributes which might be complex depending on the clients. Also there can be a large set of data summaries. So, storing the data in a relational database will be beneficial. In this case, Block is a persistent object. Also the last modification times and versions of blockchains should be kept which makes BlockChain persistent. On the other hand, some clients will have accounts which are needed to reach authenticated data. Since there may be so many clients using that data, the account information will also be kept in a relational database.

3.5.Access Control and Security

Normally, the system does not require authentication, and since the architecture is de-centralized, there is not an access control on data. However in the case of authenticated data, system requires the clients to log in by using their usernames and passwords. There will also be encryption on authenticated data for avoiding third party access. Since there is only one actor in the system, the authentication of it is controlled by a capability list. They can query data from the chains and add new blocks as they want. They will also have access to functions of NetworkManager. For receiveData() and sendQuery() functions, authenticated data should be checked by the system.

3.6.Global software control

The CrypDist system architecture is to have a decentralized, explicit software control. To respond users' quality demands and to minimize the problems followed by the distributed nature of the system, CrypDist base its content, processing, and collaboration distribution activities on the event control flow paradigm.

To process the composite events in the system such as upload of modified raw data or query data, careful placement of event-services must be done to with minimum resource usage and minimum delay based on replication and load balancing techniques [1]. Since CrypDist consists of interconnected nodes with similar tasks and functions to execute as in peer-to-peer system, exchange of events between pairs is provided. Beside the event-driven control, the content stored in the blockchain will be updated dynamically on panels on the main screen of the user.

3.7.Boundary conditions

- **Start-up:** The system is started by initializing blockchain services. To interact with user, system provides a main screen to user.

- **Shut-down:** The system will only be terminated in case of maintenance or update of servers. User interface and services on local systems will be terminated when user chooses to.

- **Exception handling:**

  - **A hardware failure:** One or more server that store raw data may fail. In case of such a failure, the system automatically detects and suspends problematic data centers or servers.

  - **Network outage:** In case of a network outages between pairs, followings are the cases:

    - The blockchain synchronization on local servers may fail. If any peer may upload a new file or update file, synchronization among peers may not be provided in a short time. The system detects the situation and protects the current state.

    - Replication of raw data among distributed servers may fail. The system detects and protects the current state.

  - **Power outage:** Power outages may occur in one or more server. In case of such a case, the system operations continue via other servers. And nonfunctional server will be restarted after power failure.

  - **Data corruption:** Due to inconsistencies in transmission, data might be corrupted and corrupted data might be uploaded to system. In case of such situation via replication of data on distributed servers and version control service, the system may recover the data.

  - **Monitor blockchain content:** The blockchain content on the main screen cannot be uploaded automatically.

  - **Data Entry:** The system fails when the user is entering information.

  - **Logging out:** The user is unable to logout.

  - **Logging in:**

- ‣ **User errors:** Displays an appropriate error message to user if any below case occurs.
    - ‣ Username is not matched.
    - ‣ Password is not matched.
    - ‣ Length of the input to username text field is lower than the allowed length.
    - ‣ Length of the input to password text field is lower than the allowed length.
    - ‣ Username text field contains unrecognized characters.
    - ‣ Password text field contains unrecognized characters.
- ‣ **System errors:**
    - ‣ The authenticated main screen is not displayed after logging in.

4. Subsystem Services

4.1.Graphical User Interface Subsystem

- Takes user input.
- Passes user input to controlling server.
- Displays system messages.
- Displays BlockChain content.

4.2.BlockChain Subsystem

- Receives the locally stored data.
- Synchronizes the BlockChain data among peers.
- Updates the locally stored data via synchronization.
- Sends data summary information Mass Data Storage Subsystem.

- Sends requests to Mass Data Storage Subsystem to upload or update data.
- Sends BlockChain content to User Interface Subsystem.

4.3.Local Data Storage Subsystem

- Stores the BlockChain data in local databases of users.
- Receives update requests from BlockChain subsystem.
- Sends BlockChain data information to requesting controllers.

4.4.Mass Data Storage Subsystem

- Stores the raw data.
- Allows authentication to download the confidential data
- Receives update, upload requests from BlockChain subsystem.
- Sends raw data to requesting controllers.

Subsystems can be seen in a detailed way in Diagram 2.

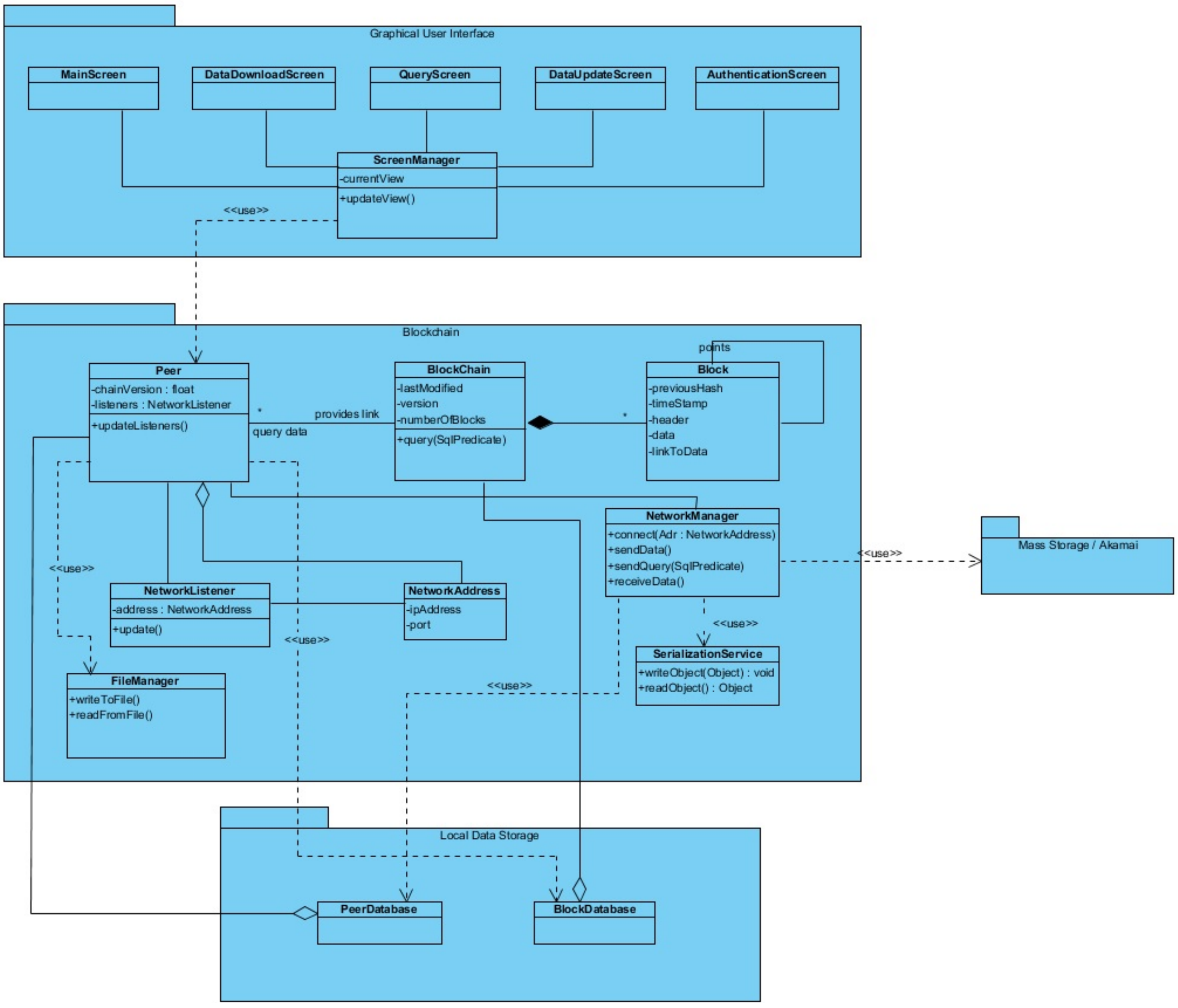

Diagram 2: Subsystems

5. Glossary

Genotype: the particular type and arrangement of genes that each organism has

Phenotype: the physical characteristics of something living, especially those characteristics that can be seen.

Blockchain: A blockchain — originally block chain — is a distributed database that maintains a continuously-growing list of ordered records called blocks.

RAM: Random Access Memory

CPU: Central Processing Unit

Java: general-purpose computer programming language

Java Virtual Machine: an abstract computing machine that enables a computer to run a Java program.

Akamai Technologies: is an American content delivery network (CDN) and cloud services provider

NetStorage: is a highly reliable, feature-rich storage solution.

dbGaP: The database of Genotypes and Phenotypes (dbGaP) was developed to archive and distribute the data and results from studies that have investigated the interaction of genotype and phenotype in Humans.

Wellcome Trust: is a biomedical research charity

Bitcoin: is a cryptocurrency and a payment system.

Subsystem Decomposition

GUI Subsystem: The user interface subsystem giving the control of the application to the user.

BlockChain Subsystem: A data management system. It is going to synchronize the summary data among clients.

Local Data Storage Subsystem: stores the blockchain data in local computer

Mass Data Storage Subsystem: stores the row data in server.

6. References

[1] . Xhafa, F., Paniagua, C., Barolli, L., Caball´e, S.: A Parallel Grid-based Implementation for Real Time Processing of Event Log Data in Collaborative Applications. Int. J. Web and Grid Services, IJWGS 6(2) (2010) (in press)

[2] https://www.akamai.com, access date 18.12.2016

[3] https://www.wikipedia.org, access date 27.12.2016

[4] http://dictionary.cambridge.org, access date 27.12.2016