DECENTRALIZED FUNCTION AS A SERVICE

Siyane Koralege Nuwan Uditha Tissera.

209386K

Degree of Master of Science/ Master of Engineering

Department of Computer Science Engineering

University of Moratuwa

Sri Lanka

March 2022

DECLARATION

I declare that this is my own work and this thesis/dissertation2 does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa, the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date:

The supervisor/s should certify the thesis/dissertation with the following declaration.

The above candidate has carried out research for the Masters under my supervision.

Name of the supervisor:

Signature of the supervisor: Date :

ABSTRACT

The objective of this research is to implement an automated, user-oriented, decentralized function as a service provider that replaces the existing centralized, single-authority FaaS providers in a way that can address the weaknesses in the global cloud infrastructure related to the serverless architecture. This refers to applications which heavily depends on third-party services running on the most well-known vendor temporary containers (or FaaS). Existing serverless architecture suffers from deficiencies such as vendor control, multi-tenancy issues, vendor lock-in, security issues, lack of monitoring tools, difficulty managing granular activity, and architectural complexity. The proposed system decentralizes granular functions across a peer-to-peer network to provide decentralized FaaS. A granular function is an atomic function with a single responsibility deployed on the Orb network. "Orb" is a peer-to-peer network of peers (nodes) and super peers (Supernodes/Trackers). Node detection on the network is done using the principles of "Satoshi Client Node Discovery". The user can register to the network as a node or super node by installing the client or server software as required. A node (represents a personal computer) provides the functionality of executing, deploying, and hosting functions. Supernodes keep a record of peers, live Supernodes, host particulate functions, and more. The user can apply an atomic particle function to the network through the server software. For deployment, users are charged in cryptocurrency (Ether). The payment form of "Orb" is based on ether smart contracts that use blockchain technology. The deployed function is sent to the Supernode and distributed across a network of peers across the network to achieve enhancement, reliability, and integrity. The confidentiality of a deployed function (a piece of code) is achieved through technologies such as private-key public-key encryption, proper obfuscation, and containerization. Users can restrict access to the function by keeping it private and opening it to a select group of users. For hosting a function, the user is paid in "ether" depending on the number of requests served and the uptime. When analyzing the response time by calling the function against an active set of peers will provide the real time analytics data about the availability and reliability. In the long run, “Orb” might support decentralized APIs.

ACKNOWLEDGEMENT

I would like to express my heartfelt appreciation and gratitude to Dr. Indika Perera, for mentoring me in the MSc Research Project as it's supervisor and for guiding me throughout the semesters.

I am very grateful for his significant assistance in the research endeavor, His assistants in this research endeavor became a key success factor, which facilitated me with the necessary skills, resources, direction, supervision, and honest feedback. I was able to complete my coursework properly in a timely manner thanks to his expertise and constant guidance. I would like to thank all of my colleagues for their assistance and support in locating relevant study material. I am really grateful for the support of my parents, brothers, sister, nephew, niece, and close friends.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
DHT	Distributed Hash Table
FAAS	Function as a Service
DFAAS	Decentralized Function as a Service
AWS	Amazon Web Services
JSON	JavaScript Object Notation
API	Application programming interface
LB	Load balancing/ Load balancer
S3	Amazon Simple Storage Service
EC2	Amazon Elastic Compute Cloud
EFS	Amazon Elastic File System
GCP	Google Cloud Project
DDoS	Denial-of-service attack
AI	Artificial intelligence
SaaS	Software as a service
IPFS	InterPlanetary File System
HTTP	Hypertext Transfer Protocol
SMT	Smart media token
DCS	Decentralized cloud storage
REST	Representational state transfer
URL	Universal resource locator
UI	User interface
MVP	Minimum viable product

CHAPTER 1

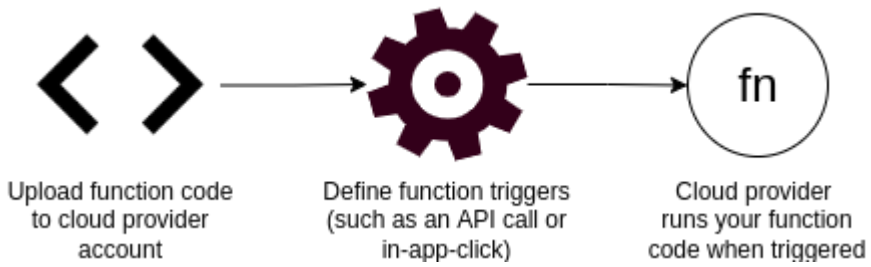
INTRODUCTION

1.1 Overview

Severless architecture is a trending approach of designing softwares that enables developers to compile and execute services without worrying about the backend infrastructure. Cloud providers have come up with services to manage these backends as a service to its users. Hence, users don’t need to worry about provisioning and scaling of storages, databases or application.

One of the most popular serverless architecture model available right now is Function as a Service (FaaS). Here developers write code in form of set of discrete functions. Each function is single responsible and will be triggered by an event specified. For example, User can write a function that sends an email once new file is added to Google drive.

This way user (or developer) only needs to focus on the code implementation and deployment. Everything will be handled by the cloud provider seamlessly.



AWS was the first cloud provider to introduce FaaS with their mainstream services. It was in 2014, known as AWS Lambda and its the most popular FaaS as of now. Currently all cloud leading cloud providers like Google, Azure (By Microsoft) have FaaS offerings namely Google Cloud Functions (GCF) & Azure Functions respectively.

Despite its usefulness, the FaaS model has a few drawbacks as well. There are some challenges developers are experiencing with continuous usage of these FaaS models.

Vendor Lock-In is a serious issue in this regard. Although cloud providers offer seamless ways of integrating with services such as compute engines, databases, messaging queues, load balancers etc.; Once a user integrates with these services it's hard to get the way out. Apart from that function code is on the control of cloud provider. Hence, despite its benefits of managed services, it has its own disadvantages too. This is termed as Vendor control.

Security is also another concern because cloud providers may run the code inside a shared server or in a server which is not configured properly. In this kind of scenario, application code is at a risk.

This research is to address the existing drawbacks in serverless computing by replacing this with decentralized function as a service platform which doesn't have a single & central authority, vendor control & vendor Lock-in etc. Decentralized system will give back the authority and ownership of data back to the system or its users. To facilitate the payment model, rewarding system with cryptocurrencies are introduced.

1.2 Motivation

This research depicts decentralized function as a system (DFaaS) could outperform a centralized, single authority function as a system (FaaS). This decentralized system will give the authority and power back to the user. This is operating as an open & crowd sourced decentralized system which provides not only the functionality of a typical FaaS but more than that.

This research introduces payment model based on crypto-currencies. Not like in FaaS; Apart from hosting function, function owners can earn through the system. Function owners are getting rewarded by other users in the system. Functions within the system are sharable and reusable, but consumers should pay for it per usage (per request call).

1.3 Purpose

The purpose of this research is to eliminate drawback in existing centralized function as a service model with decentralized function as a service solution called "Orb". This is a Unique research area that changes the definition of serverless computing. Today most systems are centralized and have a single authority. Large-scale cloud services providers will decide how data is handled and controlled along with cost.

Idle computing power is freely available worldwide. But there is no adequate mechanism to use computing power to perform calculations. By allowing each node to execute a function(s) and exposing them it could be used to utilize this computing power.

The "Orb" focuses on challenges such as implementation of decentralized services, network infrastructure, function as a service Provide, Payment and Security Model. The decentralized function as a service (or DFaaS) solution is a unique research area related to data communications that is changing the definition of serverless computing.

Currently, most systems are centralized with single authority. These companies decide how the data is manipulated, loaded and controlled. This system will benefit users by giving them control of their and power to information back to them.

This serverless architecture can be used with other architectures like micoserversices. For an example, Part of an application could be written as microservices and other part as serverless code. But that application can be considered serverless in whole.

1.4 Conceptual idea

Original idea of decentralizing function as a service initiated with the usage of existing function as a service model. When analyzing it’s drawbacks, idea of decentralizing could solve most of its own issues. Apart from decentralizing, this was initiated with the idea of added benefits.

This discussed model facilitates both developers and non-devolopers. There is around 45 million developers in the world and altogether there is 5 billion internet users

(representing more than 50% of world population). Since this model can facilitate both parties, this has the potential of growing to the level of catering world population. Rewarding system facilitates all its users equally encouraging function writers and consumers. Cryptocurrency based rewarding model along with a smart contract makes the revenue model more transparent to it's users. This attractive model of rewarding is a key factor to drive this solution to global.

Rewarding model means to be based on Ether or Etherium. Reason to select Ether is it's features and compatibility on smart contracts. Despite smart contracts, Etherium the most popular blockchain on dApp and DeFi solutions.

This solution is designed to be open-source and crowd-funded. Initial design of the system will be done as MVP (Minimum viable product) and can be released to alpha and beta testers. After few testing grounds, this can be released to GA (General availability).

1.5 Research problem

Decentralized FaaS is a new concept that did not exist until now. None of the service providers that exist today offer a decentralized solution despite the centralized FaaS. Although some service providers have enough server space to implement our application, most of them work centrally.

As far as the FaaS is concerned, there is no suitable way or existing systems that use the decentralized FaaS concept from today. The reason someone would decentralize a function is to avoid any central points of failure. Developers can create, deploy, and monetize serverless features completely anonymously and users can consume them in complete privacy.

In a typical environment, traffic can be intercepted through a single point using a load balancer and routed to different servers. In this system, Supernodes act as a LB(Load balancer) itself. Requests (or function calls) are passed through Supernodes based on their measured latency.

The attractive feature whereby a user would host a function on their own machine that will make the node eligible for payment through an Ethereum contract. The payment model works as of a configured contract. More traffic will generate more money.

The interesting part in this model is that the user himself can host and deploy at the same time. So the cost of implementation can be prevented by hosting and contributing more towards network growth.

The user will use "Orb" instead of uploading it to AWS or any other because decentralization reduces costs, makes the owner completely anonymous, all transactions are verified and more user contributions to the network are paid in the form of Ether. Data ownership and central authority are completely avoided with this approach.

In the end, the user receives a reward that will be proportional to the contributions made to the system. Decentralization has its drawbacks. Once deployed, how can you modify the owner? How do you decommission the function? In a decentralized environment, it is difficult to update functions.

Updates should be managed as versions of that feature and the tracker always chooses the latest version of the requested function. The older version will get deprecated once the user issues a request.

What are the security measures on the nodes of the system? By default, the system does not have a central point of failure, which makes the system fully tolerant to DDoS (distributed denial of service) attacks.

The system is also resistant to intermediate attacks as function calls will be redirected through Supernodes. Every time, users can call 2 different nodes in 2 different locations avoiding one route.

Only one executable of the function will be implemented (without the source code) as a safety mechanism. To ensure that decentralized functions remain unchanged, Supernodes will be checked automatically by comparing a dynamically generated hash and the existing hash, each time a pair is triggered.

1.6 Research objectives

Research objective concludes the components that is going to achieve at the end. This research contains many objectives. The purpose of these research objectives is to achieve the goals in the research in a guided path. Research objectives helps to narrow-down the focus to each topic in a divide and conquer approach.

FaaS helps developers to run code without paying attention to the server infrastructure. But since these severless services are managed by cloud vendors, it has its own pros and cons. Vendor control, multiple ownership issues, vendor lock-in, security concerns, lack of debugging and monitoring tools, difficulty managing granular functionality, and architectural complexity are some of the downsides.

This system solves vendor control, multiple ownership, vendor locking and security issues by decentralizing granular functions over a peer-to-peer network to provide decentralized FaaS. A granular function refers to an atomic function with a single responsibility (such as add function, image recognition, IoT connector) hosted on the network. "Orb" has 5 major challenges as below.

- Decentralized service deployment
- Network infrastructure
- Function as a service model
- Payments model
- Security

An introduction to the major challenges is given below.

1.6.1 Decentralized Service Deployment

After developing a certain service (API, Lambda function), the user must host it to be able to use it. Since complex applications can consume more computational power and the cost can be high, the developers or owners of the application may tend to host it on the traditional centralized cloud computing platforms such as Amazon Web Services. There may be outages [11] as all centralized systems come with a single

point of failure [12]. The most recent outage is the "AWS S3" [1] outage where thousands of websites suffered downtime. This solution may be advantageous since the only point of failure [12] is eliminated. In existing cloud computing platforms such as AWS EC2 and GCP (Google Cloud Platform), user will be billed for disk space, memory, number of CPU cores, and more importantly, the code implemented in the above providers will be theirs. The decentralization features lead to the elimination of exclusive powers and reduce costs.

1.6.2 Network Infrastructure

Torrent technology has been around from 2001 to the present. It is not completely P2P because centralized servers called “trackers” are used to keep track of nodes and files. When a torrent client requests a file via trackers, the relevant set of pairs will be given, and the download begins. When this happens, the user can enable DHT (Distributed Hash Table) that will keep track of the pairs live. Now the torrent can request a file fragment from peers and continue the download even without the Tracking server depending solely on peers. The downside to torrents is that you need a centralized server if you use a tracker to identify peers. If the stream is dependent only on DHT to find pairs, at first the client needs to know about a live pair to get started and then DHTs will be updated, and the network will grow. This would be the ideal technology for the proposed infrastructure. The proposal of the system is not to split files like in torrents, but the distribution of functions should be handled similarly to the BitTorrent protocol. However, the challenge is to implement the BitTorrent protocol so that it adapts to the "Orb". A network of trackers (super nodes) will help users to locate the implemented functions and monitor those as a gateway.

1.6.3 Function as a service model

Function as a service model can be treated as a cloud computing model that enables users to implement, execute, maintain and manage granular functions without any

hassle of keeping servers dedicated for this facility [2]. This way users can achieve serverless nature in an application [4] and this helps on implementing microservice based applications [3]. Serverless does not mean "no servers", but server mapping. The resources will be administered and managed automatically by the service provider. In this system, the user does not have to configure server resources like memory storage. User only have to implement the function, test it and deploy. After that new endpoint will be generated from the cloud provider and that can be shipped. The basic infrastructure for hosting is a p2p network [23]. Since this functions are deployed in a decentralized network, functions cannot be edited or easily removed due to the decentralized nature of the system.

1.6.4 Payments model

The purpose of the "Rewards System" in "Orb" is simply to provide users who act as sowers, owners and applicants a payment model that acts logically to provide fair rewards to all. Orb's key uniqueness is the rewarding component. "Orb" provides the reusability of the serverless functions and most importantly a revenue generation through the implementation of features and hosting. The rewards are calculated according to a combined relationship derived from the average time complexity and the complexity of the code of a function. The owner of the function that implements the useful serverless role will allow your role to be distributed across the "Orb" nodes. Further away Hosted nodes and role owners will be rewarded for each request called by an Applicant and applicant will be charged for using it. Although there are services that provide similar services to “Orb”, None of them have a rewarding model that allows users to make real profit from it.

1.6.5 Security

The decentralized nature of the system makes the "Orb" resistant to DDoS (Distributed Denial of service). Supernodes to route requests when a client is consuming a hosted

function makes the system more robust and man-resistant in intermediate attacks as the path to the consumed function will only be decided by the Supernodes. Most of the financial models that exist today are based on trust. "Orb" brings an unprecedented level of trust to the users by eliminating a third party its payment model. Users will initially subscribe to an Ethereum contract where the need for trust is eliminated. As far as privacy is concerned, the functions will be anonymous in the system. A function you will only have a unique hash to access it. Only the user who implements this knows the property of the function. To make the hosted features tamper-proof, The executable will be implemented (without source code). If the functions are developed as executables, they could still be reverse engineered. To avoid this, "Orb" uses a hashing system where the function is initially executed through a hash and that hash is distributed across the Supernodes. When a node connects to a Supernode Initially, all hosted functions will be validated before exposing the function for consumption.

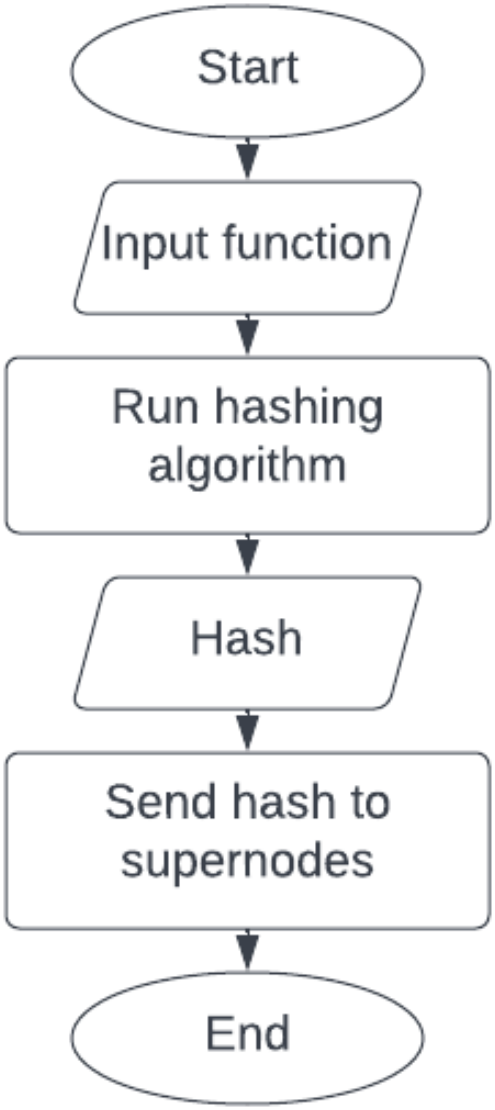


Figure 3.1 Hashing function

CHAPTER 2

LITERATURE REVIEW

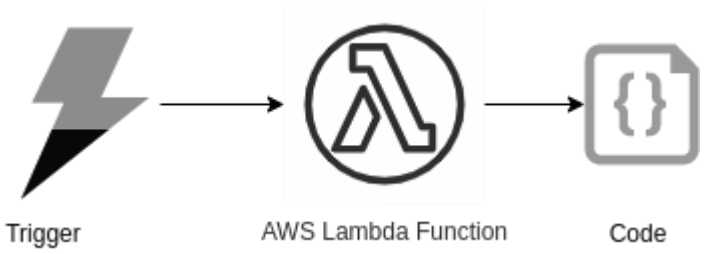
2.1 Overview

This chapter is focussed on identifying potential similar research in this space with latest implementations built around serveless architectures. Serverless architecture is a trending topic in the world and it was greatly evolving for past few years. Function as a service was built on top of serverless architecture to facilitate event based compute services which are provisioned and managed as a service by cloud providers. Other popular implementations like decentralized storage services, decentralized social networks, distributed file systems which are built on top of peer to peer architecture are also discussed & evaluated in this literature review.

2.2 Serveless architecture providers

2.2.1 AWS Lambda

AWS Lambda is a server-less, event-based compute service from world famous cloud service provider Amazon Web Services that provides server space for functions to be deployed. Using this service, we can run any code that represents the back-end of an application or service with a zero administration.



AWS Lambda can be integrated with more than 200 of other AWS services and pay only for the usage. AWS Lambda uses high capacity computer infrastructure to run and manage our code. This includes high end computer resources including compute engines, operating systems. AWS Lambda also facilitates features like automatic scaling of compute engines, Code monitoring tools etc.

2.2.1.1 AWS Lambda Benefits

In AWS Lambda, You can run the code without providing or managing the infrastructure. User only needs to write the code, zip it and upload. Then it deploys automatically in a way that it can responds to requests on any rate. It can basically handle millions of events without a hassle. Instead of providing infrastructure for maximum capacity, here you can save money by paying only for the calculation time you use per second. AWS Lambda optimizes code execution time and performance with the correct amount of active memory. It responds to the highest bidder. AWS Lamda evaluates double digits milliseconds with provisioned concurrency.

2.2.1.2 AWS Lambda by examples

2.2.1.2.1 File processing

Amazon S3 (Amazon Simple Storage Service) can be configured in a way to process files uploaded to S3 using AWS Lambda processing. Existing Amazon EFS file system also can be enabled to configured for large-scale parallel access file processing.

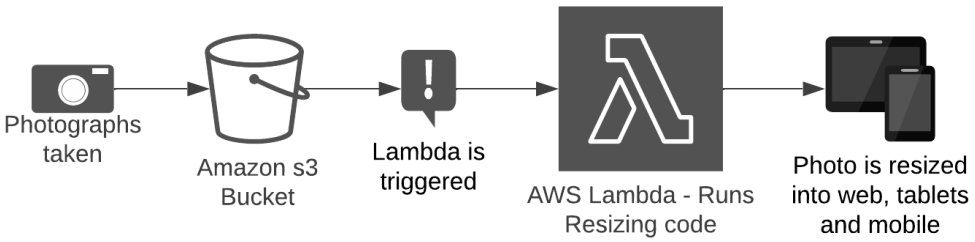


Figure 4.1 File processing in AWS Lambda

2.2.1.2.2 Stream processing

AWS Lambda & Kinesis can be configured to process & filter real time streaming data. This can be used for tracking purposes, activity logging, order management, and other streaming activities.

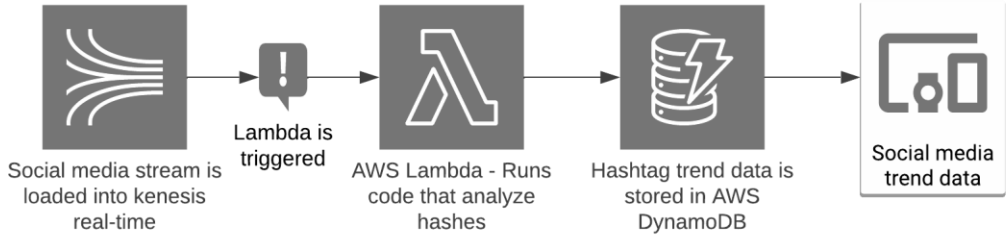


Figure 4.2 Stream Processing in AWS Lambda

2.2.2 Azure Functions

2.2.2.1 Overview

Azure Functions was deovloped in 2016 to comete with AWS Lambda (which was established on 2016). Azure Functions offer similar kind of functionality and its focussed on elaborating Microsoft or Azure services along with Azure Functions.

Azure Functions enables users to execute a trigger based script. Infrastructure is managed by Azure as a managed service. This is used to achieve decoupling in production systems. This can also be used to achieve high throughput.

Azure Functions have gained popularity because it is,

- Completely serverless
- Lightweight
- Easy to write and deploy
- Fast to execute
- Compute on demand
- Muti language support
- Zero maintenance
- Browser based testing and deployment

2.2.2.2 Azure Web Jobs

Azure Web Job is a piece of code that runs on top of Azure app services. Azure Functions are built on top on Azure Web Job with added features. Azure Web Jobs are 2 types.

- Triggered Web Jobs
- Continuos Web Jobs

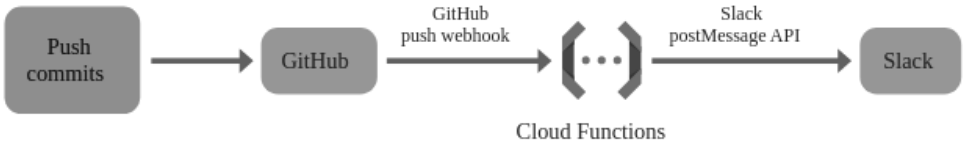
	Azure Functions	**Web Jobs**
Trigger	Azure Functions can be triggered with any of the configured trigger, but it doesn't run continuously	Web Jobs is of two types, Triggered Web Jobs and Continuous Web Jobs
Supported languages	Azure Functions support various languages like C#, F#, JavaScript, node.js and more.	Web Jobs also support various languages like C#, F#, JavaScript and more.
Deployment	Azure Functions is a separate App Service that run in the App Service Plan	Web Jobs run as a background service for the App services like Web App, API Apps and mobile Apps

2.2.3 Google Cloud Functions

Google Clould Functions is also another leading Function as a system model (FaaS) offered by popular cloud vendor Google. Google released the first version of GCF in 2017. GCF was behind the competition on the early 2018, but have been managed perform better now. But still, AWS Lambda is leading the FaaS space.

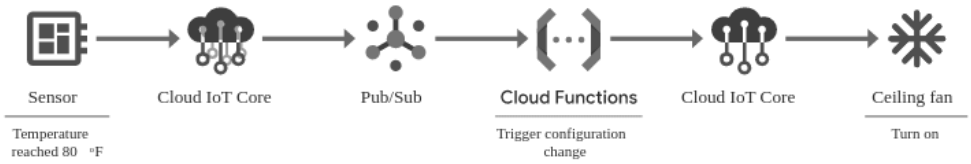
Functions can be setup in a workflow to run. This can be configured as a series of set of events too. Cloud functions can be invoked using HTTP listener, Google Cloud Storage, Google Pub/Sub, Cloud Firebase, Firestore etc. GCF is a scalable and comes with pay-as-you-go payment model.

Google have come up with many use cases to showcase the functionality of Google Cloud Functions and Google have the potential to outperform in near future. The following use case explains an integration with Github and Slack. Here Github has a webhook for the push events and cloud function emit the specific event as a Slack message.

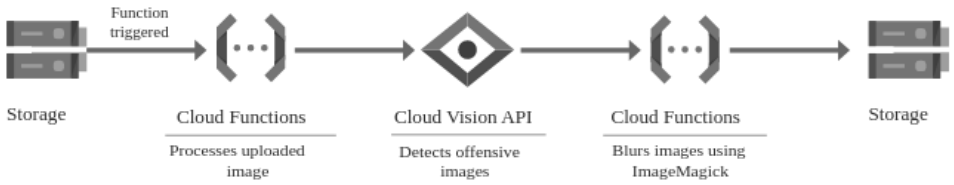


Google Cloud Functions also can be used with Google Firebase to facilitate the functionality of a mobile backend. This can handle authentication, user logins and event based notifications too.

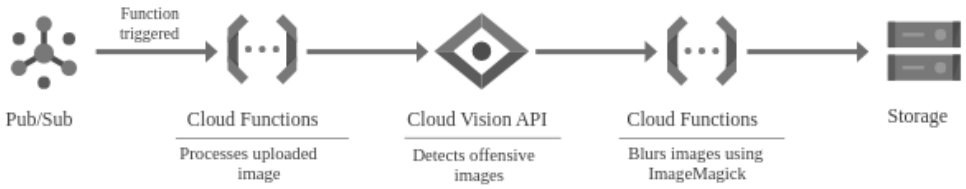
Using Google Cloud Functions with Google IoT core can facilitate real time processing of IoT data. Cloud functions can be used to write custom logics for events that occur or sensor changes.



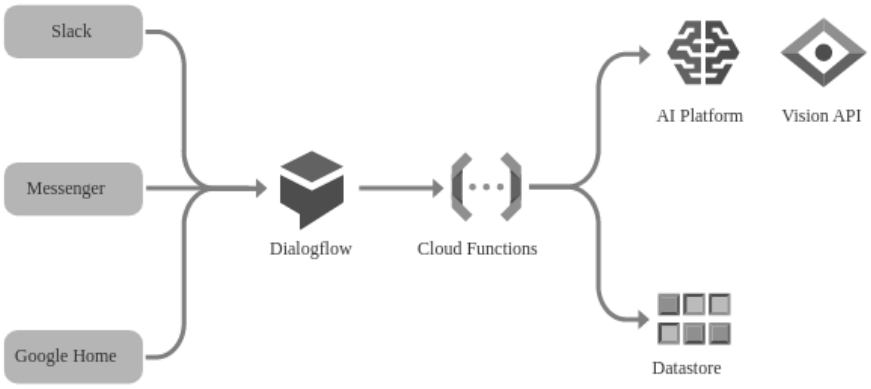
Real time file processing can be facilitated with the usage of Google Cloud Functions and Vision API. File upload event can trigger cloud functions to analyze images and Cloud API can detect offensive images. Then these offensive images will be blurred using a third party tool. This action is also invoked by Cloud functions upon detecting blurred images. Then theses modified images can be saves again on the specified storage.



Google Pub/Sub is used to detect realtime transaction processing data in data analytics. So Google Pub/Sub in conjunction with Google Cloud Functions can be used for analysis of real time information of social networks etc.

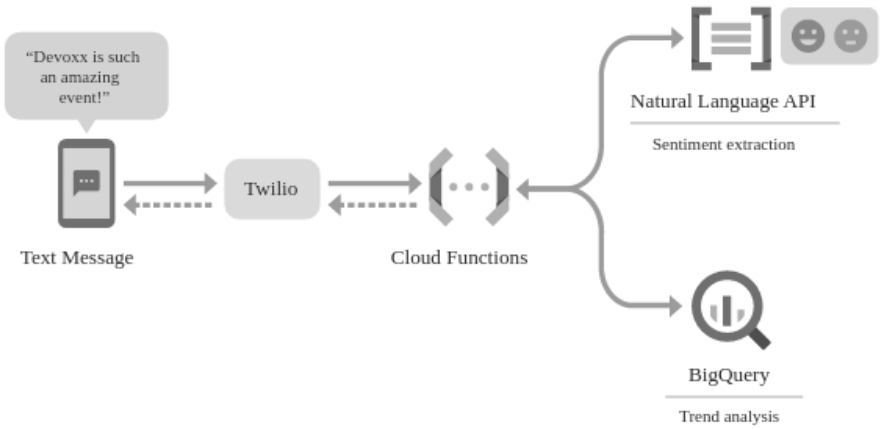


Chat bots or virtual assistants are widely used in the current systems to bring artificial intelligence to application that doesn't have powerful backends. For example, Dialogflow & Google speech API can be used with Google Cloud Functions to bring automated chat experience to any application.



Sentiment analysis is one of the usefull integration in cloud. Sentiment analysis is identifying the emotional aspect of a sentence or text. This can be used on chat bots to

identify user's emotions while texting. The following is an example of using Twilio trigger with Cloud functions to analyze messages.

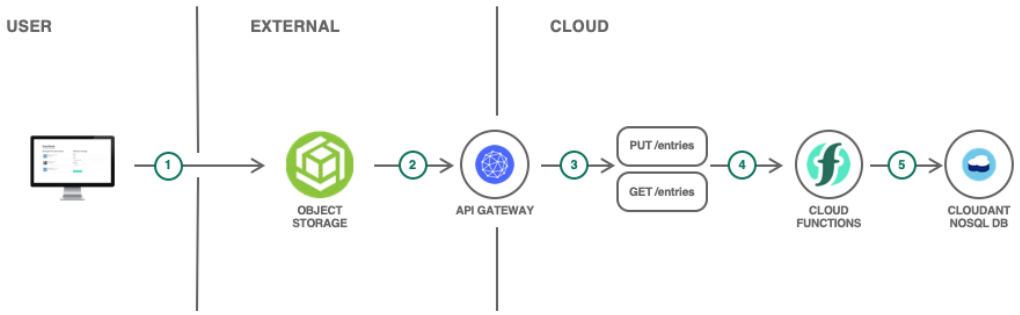


2.2.4 IBM Cloud Functions

IBM Cloud Functions is the latest cloud provider on FaaS space. There is a special case in IBM Cloud Functions; No time limit for single invocation of a function. Concurrency is a question in IBM. As per there documentation [43], "Two actions can run concurrently and their side effects can be interleaved."

IBM Cloud Functions is only a managed infrastructure for opensource Apache OpenWhisk [44]. If someone is looking for an opensource solution Apache OpenWhisk is a good option.

The following diagram [45] is an example of using IBM Cloud Function to implement a system that writes to a NoSQL database when user post messages. Here, user access the application which is inside a object storage. That application executes API calls which hits on the API gateway to trigger the cloud functions.



2.3 Distributed storage

2.3.1 IPFS

2.3.1.1 IPFS overview

IPFS [13] is a fully distributed system built to store files. It maintains a connected file system across all devices the network.

IPFS [13] is a versioned distributed file system that can grab files, manage, store them somewhere and then track the versions over time. IPFS [13] also takes into account how those files move across the network so it is also a distributed file system. This file system layer provides some very interesting features, such as:

- Fully distributed websites
- Websites that do not have an origin server

The goal of IPFS is to surpass HTTP to build a better website with targeted objectives. IPFS have addressed the drawbacks of the current web just as,

- Expensive & inefficient.
    - HTTP downloads files from one server at a time - but peer-to-peer IPFS captures multiple nodes at once, enabling significant bandwidth savings. With savings of up to 60%, IPFS can efficiently distribute large amounts of data without copying.

- Originates from its backbone.
    - IPFS empowers the creation of various distributed elastic networks that enable the Internet spine to survive - with or without a connection.
- Centralized with less opportunities.

2.3.1.2 IPFS Content Addressing

IPFS refers to all of its objects by hash. Objects include images, videos & articles. Normally in other systems objects are referred by the server which it resides. When user wants to access a file, IPFS will ask for that file from the entire network. IPFS will check entire network and find where that hash is located and return the location to access that file. In this way content addressing is used by IPFS at HTTP layer.

2.3.1.2.1 IPFS Objects

IPFS is sharing IPFS objects in a P2P system. The IPFS object has 2 fields. It is a kind of a data structure with Data and Links. Data is a field less than 256kB that contains a Binary large object or BLOB of unstructured data.

Link structure consiste of 3 data fields:

- Name on the link
- Hash of the IPFS object
- Size of the IPFS object

These links maintains relationship between IPFS objects.

2.3.2 Storj

2.3.2.1 Storj overview

Storj is a decentralized platform to persist data securely. Payments are done by cryptocurrency known as “Storj” which is unique to them. Storj enables storage options for users which is cheaper than normal storage vendors. But this is faster and safer than other storage vendors in this space.

In a traditional storage mechanism, files are served from a one location like from a server or shared network device. In here, files are served from multiple locations. So this is very much faster than a traditional storage mechanism. If one location is incapable of serving the file or having a high latency, that source can be eliminated and dependent on another location where that file is available.

Client is able to collect and merge counterparts to view it as a single file to the user. This operation can be performed quicker (with lower latency) than retrieving a file from traditional http servers.

Storj is cheaper than traditional storage options because this uses free disk spaces of users who are willing to rent. So rather than paying for a dedicated server with 24/7 uptime, here we are renting free spaces availble on personal machines.

Users who are renting the storage spaces are also getting a considerably good about of price for their free hard disk space which is a really good investment for them.

Storj encrypts data before storing data on other clients machine. This way it ensures security of data for the end user. It also uses some advanced private & public key encryptions too. Additionally they use cryptographic hashing of functions.

Storj doesnt have any central servers. So your data is not in hands of a third party.

2.3.2.2 Storj Decentralized Cloud Storage (DCS)

Storj provides secured, private decentralized data storage. It outperforms traditional typical cloud storage systems in this space. Storj is affordable and predictable on pricing. At the same time its compatible with S3. Storj provides open source documentation support too.

Storj is predictable when it comes to pricing. User can predict the price per capacity. The free plan (Get started plan) comes free of charge. This can be used for anytime with a limited capacity.

Free plan consists of storage 150 GB per month (with limited bandwidth of 150 GB). In Pro plan user have to pay for excess usage exceeding 150 GB. This is good for companies and large projects that needs more space. For every GB passing 150 GB, user have to pay 4 USD. (Cost for bandwidth is 7 USD). This way Storj is affordable and predictable.

Storj is having zero confident, high available, multi-zone architecture. Although being open-source, users have full ownership & control of their data.

When a user upload a file to Storj, Its generally splits into 80 counterparts (or more) and this is distributed across 1000's of nodes in a very short time. These nodes will be spread across countries that covers the whole world. Normally its distributing to countries around first.

In this way, File can be regenerated from another node by merging all the counterparts available. As per Storj they always keep more than 30 ways to generate our file in a situation of data breach or crash on the network. This way it assures more level on availability & partition tolerance.

2.3.2.3 Benefits of Storj DCS

Storj provides more benefits users and as well as devolopers. Being open-source, community driven and flexible devolopers are more satisfied in Storj DCS.

1. **Availability**

Files are distributed among thousand of nodes which are available in different countries. As per StorJ there have more than 30 ways to regenerate a files on retrival.

2. **Safety**

Safety is assured by storing files in a distributed, but secure and private DCS. Being private, no one can tamper things there easily. Storj provides good authentication and authorization on the system too.

3. **Durability**

Storj automatically repairs broken files. Its an advanced feature on Storj.

4. **Privacy**

All data in Storj is encrypted using best encryption techniques. So viewing, editing or sharing is restricted in all forms without proper authentication and authorization.

5. **Cost Efficiency**

This provides the most cost efficient way of DCS in the market. This comes with a free storage of 50 GB which is quite enough for a standard entry user.

6. **Developer Friendly**

Storj provides a API that is handy for developer with improved developer tools and technical documentation. There are tutorials, videos provided with free access for anyone to follow.

7. **Open Source Freedom and Flexibility**

Storj is fully transparent to the user. Being open-source it provide freedom and flexibility of using other storage options along with this. User is not locked in the system (vendor lock-in).

8. **Community driven**

Storj is community driven. Users share the information, discuss the issues, collaborate with other developers within this community.

2.4 Decentralized social network

2.4.1 Steemit

2.4.1.1 Blockchain online social media (BOSM)

Usually people are sharing their personal information and daily activities on social networks aka online social networks (OSN). Total number of people using OSNs are 3.8 billion while total number of internet users are 4 billion. Most of the online social networks are centralized and it has several disadvantages. Privacy is the main concern.

There were data breaches in popular social networks that has adverse effects on privacy. One of the major scandal in history is data breach of 87 million users in facebook for succeeding organized political goal.

The above mentioned problems motivated centralized social networks to fall. Leading developed countries like China also have banned leading social media networks due to it's privacy concerns. Ultimately social network evolution turned towards decentalization. This led to emerge Decentralized online social networks (DOSN) which is a p2p system.

Decentralization methodologies are frequently changing and its happened for last few years. Evolution of blockchain technology was a remarkable milestone is history. Blockchain is a distributed public ledger and its shared among all the parties in the network. This blockchain grows as a chain of blocks. The first most application out of blockchain technology was Bitcoin.

Smart contracts which evolved with blockchains was so famous. Smart contract is a written piece of code which explains the agreement between parties. For example, How profit should be divided between the parties. Ethreum was built on top of blockchain specifically for smart contracts.

Decentralized online social networks had lack of attention because users had no much motivation to move there. As a forward step to increase motivation & engage more

and more people in decentralized social networks, Blockchain online social media (BOSM) was proposed.

BOSMs identify user's content (posts) as wealth in the system and reward users for each contents that is being created. Users who positively engage in reactions are also getting rewarded by this model.

Steemit is one kind of such BOSM, which has most users (more than 1 million users). Steemit is based on the Steem blockchain.

2.4.1.2 Objectives of Steemit

Steemit is a social network that rewards cryptocurrencies for content creators and content rewards platform that turns the crowd into beneficiaries of the care economy. Steem is used as a coin which is dedicated to them. They use it to reward the user for usage of social media platform. This is purely based on blockchain and first one to use on social media platform.

Steemit is a social media platform that works by making the crowd reward for their contents. It supports Steem blockchain [18] and cryptocurrency alot. Steem is 'minted' daily and distributed to content producers based on the votes they get. Plus Social networking sites extract value from their user base for the sole benefit of shareholders. Steemit is different, it is a new kind of attention economy. Connecting with the Steem blockchain (which is decentralized and crowd-controlled), Steemit users receive all the benefits and rewards for your care.

Content creators are most appreciated in this network and its a value addition. Main focus of this network is to maintain a set of potential users who are really interested on dApps and at the same time attract new users to the network with interesting rewarding model. Good rewarding model broads the network easily.

Steemit wants to reward users that publishes content and help to grow the network. This way its becoming popular more and more.

Steem (Coin used in Steemit.io platform) is built from scratch to mitigate the current existing issues in social media economy. This is why Steemit is ahead in this competition. Since social media is so popular now, dApp that collaborates with crypto will give interesting features to the users. They already have some cool tactics on getting the user on ground and keeping users caught inside. They are in a plann of have significant improvements in coming days. Its everyday evolving with feedbacks from the users.

One of there main goals in this effort is to build an algorithm to examine each ones contribution to the network. There evaluation and research on this topic is having so much of value for whole dApps community.

Steem platform performs very fast compated to other dApps existing. Performance benchmark is for transaction rate of 3 seconds (3s, 24h, 7days). This blockchain is designed to handle large scale applications including social media apps. This handles far better that what Bitcoin and Ethereum does. Since this is becoming so much popular and 1million users are already using it.

2.4.1.3 Smart media tokens (SMT)

SMT is a digital asset of Steemit. SMT's can be used to develop any platform and its encouraging the users to engage in the system more and more. SMT motivates all users to create & sell POB tokens. This is generally known as Proof-of-Brain tokens.

2.4.1.4 Price of Steem

Price of Steem is $0.387329 (per STEEM) as of now (10th May 2022). Steem is predicting to reach $ 1 in future. That depends on many factors though. Factors such as,

- New competitors on this space (Decentralized social media applications)
- Gradient of users joining the system
- User churn rate (Measure of users moving out)
- New features introducing to the system
- Technical feasibility to handle large number of growing users

2.4.1.5 Earn from Steem

Most trending platform where users sell there content to earn money. Being a social media network this is being so popular right now. Its really interesting to earn money out of social media which we are using at our leisure time.

To earn money, You need to write up good, interesting posts. So many users will engage on them, react, upvote. So each of these actions will generate money for the original user of the post. This is a really good motivation for posting creative, sharable, debatable stuff.

Steemit have user levels, starting from beginner. When you are engaging more in the system, user level goes up. This makes you earn more than previous. They call this “Voting power”.

Voting power is once eligibility to earn out of a post. This is increasing with engagements. The more you engage, the more you earn.

CHAPTER 3

DESIGN & IMPLEMENTATION

3.1 Network Design

"Orb" is a multi-component distributed system located on various machines that communicate and coordinate actions to present the end user as a single integrated system.

Machines (or compute devices) that are part of a distributed system can be computers, physical servers, virtual machines, containers, or any other node that can connect to the network, have local memory, and communicate by sending messages.

Distributed systems may sometimes be ambiguous, but they generally have three basic features: all components operate simultaneously, no global clock, and all components fail independently of each other.

This system consists of 3 major components.

- Nodes
- Supernodes (Trackers)
- Payment manager

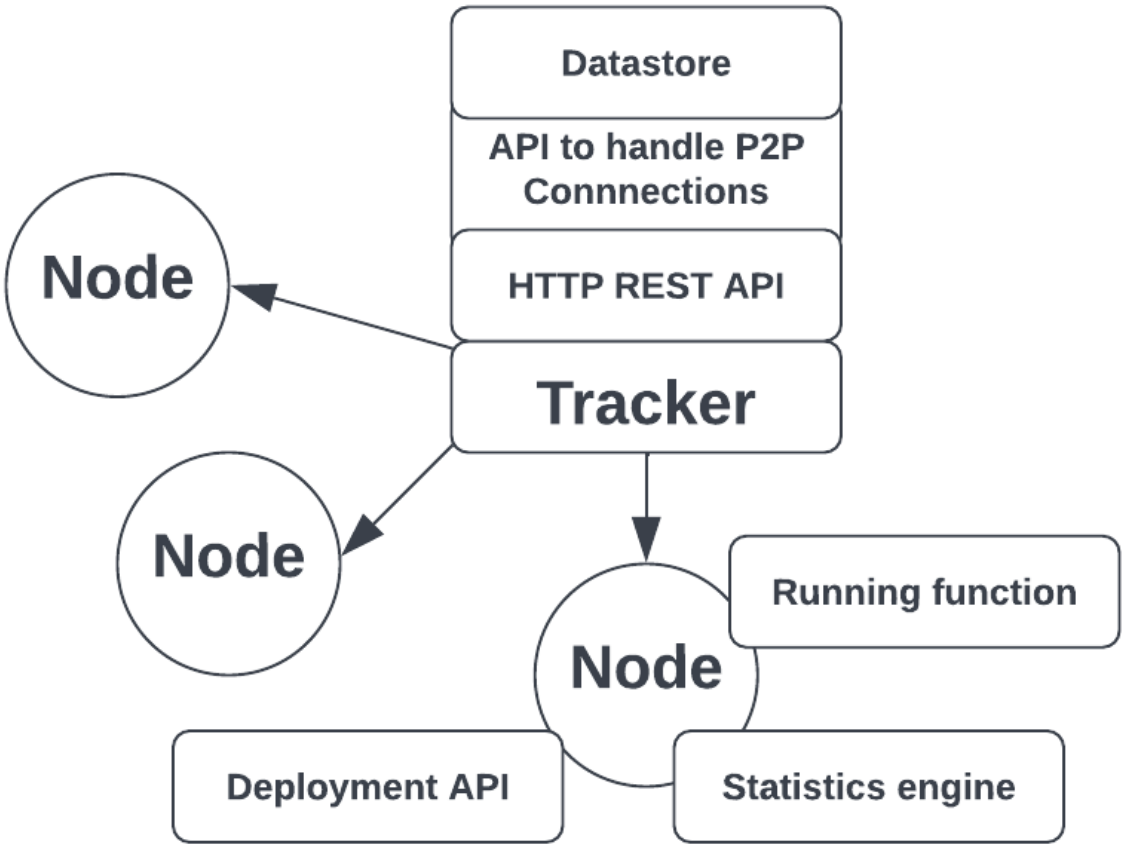


Figure 5.1 "Orb" Architecture

These components have specific tasks to perform.

Supernode or Tracker connects Nodes to it. There may be one or million of Nodes connected to one Supernode. There can be many Supernodes too. Supernode functionality is to,

1. Keep track of peers.
2. Keep track of tracking server.

Supernode contains a REST API to retrieve data via HTTP. This API provides metadata about Nodes & Supernodes in the system.

Node is always connected to a Supernode. In distributed terminologies, Node can be a physical Node or logical Node. Here it's a physical node. This can be a personal computer, mobile phone or device that can communicate with Supernodes. This can

be related as Client. Hence, we can call it a Peer too. Functions of a Node can be listed as following,

1. Keep a track of a set of peers.
2. Able to deploy a function to the network.
3. Able to call the deployed function.
4. Contains a list of hardcoded always running trackers to initiate the network.
5. Able to run and deploy functions.

3.1.1 Nodes

A node represents a personal computing device with the "Orb" client software installed.

The responsibilities of the node are as follows

I. Implement a function on the network.
II. Peer and Supernode Tracking - Using DHT (Distributed Hash Table) Servers and Super Nodes.
III. Provide URL to call a user-implemented function.
IV. Capable of executing an available function: Executing an available function on the client. The machine will generate income in terms of Ethereum for the user.
V. Subscribe to a payment model when deploying an application.
VI. A wallet to secure the payments received.

The node will consist of the following proposed mechanisms to ensure that the functions deployed are available.

I. Multiple replicas of the same function will be deployed on different nodes.
II. A tracker will automatically display the particular function when there are no asset companions available.
III. Only one executable of the implemented function will be implemented (without the source code) as a security mechanism.

To ensure that decentralized functions do not change, the Supernodes will be checked automatically by comparing a dynamic generated hash and existing hash, each time a pair is triggered. To ensure reliability and avoid a single point of failure, there will be a chain of Supernodes connected to each other.

3.1.2 Supernodes

In p2p networking, Supernode can be considered as any node within the system that can perform as a proxy server. Supernode must be capable of controlling & manipulating data flows within the network.

Supernodes keeps track of Nodes joining & leaving the network. When a node wants to join the network, It calls the Supernode for a join request. This is basically a socket communication. Supenodes actively accept new join requests and keep track of nodes attached to it. When one node leaves the system, Nodes sent a signal to the Supernode informing disconnection. Once Supernode acknowledged the leaving, Node is removed from the system. Supernode uses DHT for this tracking purpose.

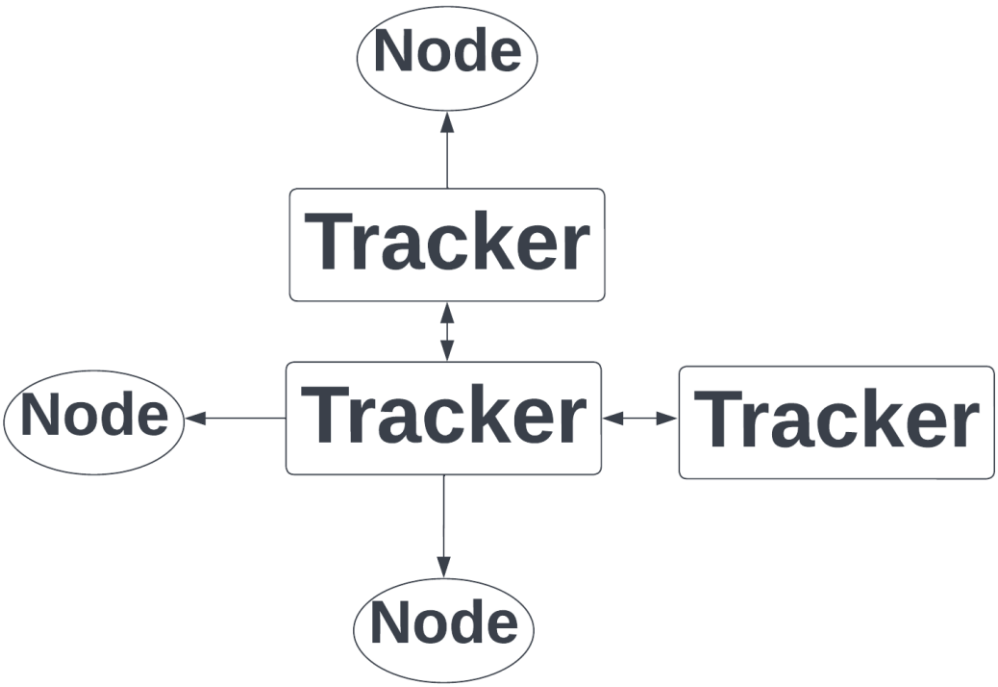


Figure 5.2 Trackers & Nodes

The responsibilities of the Supernode are as follows: Peer and Supernode Tracking API endpoint generator that generates global endpoints for implemented functions REST API (Representational State Transfer) to communicate with the outside world for data analysis. A user can choose to be a Supernode. Supernode must be in an audience network with an assigned domain name.

3.1.2.1 DHT

DHT or Distributed hash table is a storage tool in distributed systems which uses key value pairs to facilitates lookups on the network. It is a scalable and robust structure that network to find desired content host.

Each node consists on a DHT which is responsible for maintain key value pairs. This facilitates efficient retrieval process given a key in the network.

This is more like a hash table. Every data in the network is hashes and these hashed data is kept closer nodes with similar hash value.

DHT is a data structure that connects identical indicators with data. Entries (indicators, data) in DHTs are stored in pairs. You can search for data in a logarithmic overlay. DHT self-manage their behavior as nodes or services within the system can dynamically adapt to exit, connect, and fail. Recently, DHTs have been used to build Internet-sized systems that contain hundreds of thousands of components (node, service, data, and file).

DHT is a scalable storage structure that can expands from thousands to millions of nodes. Its fault tolerant design facilitates the nodes to join and reconnect any amount of times unconditionally. Being decentralized and autonomous, DHT doesn’t possess central governance.

3.1.3 Payment manager

I. Owner

In the system, the initial implementation will cost the user in terms of Ether [5]. To a particular user can implement a function. The decentralized network will decide the deployment process particular user can implement a function. In addition, the network will monitor the use of the function. If that function exceeds a particular request limit, the owner of the function has to pay all the nodes hosted through an Ethereum contract. When the user puts the requested amount of Ether to the contract, it will be distributed among nodes based on node participation.

The function owner is also a node in the system. Any node could be a function owner by deploying a function on the system. At the same time, That node can be a function user. All nodes in the system can use the functions shared in the system. They need to pay in Ether for usage.

II. Nodes

Nodes will be paid in terms of Ether. Nodes can contribute to the network & use available functions in the network.

3.1.3.1 Smart contract

Smart contracts are used in the above rewarding model. These are piece of code written inside a blockchain which can execute once conditions are satisfied. This can be configured in form on a trigger to execute a workflow.

Smart contracts follows a simple code logic of “If / When.. / Then ..”. If condition consists of the pre conditions that needs to be met. Action that needs to be executed are coded on “Then” block. This can be something like fund transfer. In our example, Its sending Ether from one account to another account. These transactions are irreversible. Once a transaction is done, it cannot be reversed.

Smart contract need to be designed and coded by programmers. Due to high demand for smart contracts, there are some apps, templates and online tools to construct smart contracts.

3.2 Implementation

3.2.1 Client studio

Client studio is the application that involves in user activity on the network. Client studio is capable of registering the user, signin, writing the code, deploying, consuming available functions in the network, paying, maintaining a wallet, etc.

ElectronJS is used to develop this client studio. This is a cross browser application that can work on any pc under any operating system. Same source code can be used to built executables that suits all available operating systems right now.

Client studio needs a supernode to connect to the network. So initially it search for supernodes and try to connect available supernodes.

main.js

```
const {app, BrowserWindow} = require('electron')
const path = require('path')

require('electron-reload')(__dirname, {
   electron: path.join(__dirname, 'node_modules', '.bin', 'electron','public')
});

const url = require('url');
const soc = require('./public/services/request.service');
const io = require('socket.io-client');

var ipcMain = require('electron').ipcMain;

global.Url ="";

looper();

function looper(){

   soc.Request();

   global.Url = soc.SocUri();

   if(!global.Url){
      console.log('searching Super node Uri' )
```

```
    setTimeout(function () {
        console.log('Req : '+soc.SocUri());
        global.Url = soc.SocUri();
        looper();
    },500);
  }else{
    console.log('loop exist : run app')
  }
```

Client studio needs a bootstrap server on start. So we need to hardcode the Uri of bootstrap server as below.

request.service.js

```
const request = require('request');
var electron_remote = require('electron').remote;// this module initiate pipe with
electron app

var SocUri = "";

const boostrapUri = "http://192.168.1.6:3300";

module.exports = {

   Request : function () {
      request(boostrapUri, function (error, response, body) {

         try {
            if(!error){
               obj = JSON.parse(body);
               var uri = "";
               var latency = 0;

               obj.forEach(data=>{
                  console.log("Online Supernodes : "+data.uri)
                  if(latency<data.latency){
                     latency = data.latency;
                     uri = data.uri;
                  }
               });

               if(uri.length>0){
```

```
                    console.log('resolve super node uri : '+uri);
                    SocUri  = uri.toString();
                }else {
                    console.log('zero super nodes')
                }

            }
        }catch (e) {
            console.log("Request Service | Request | Exception : "+e)
            console.log("check boostrap server Ip Address : trying to connect =>
"+boostrapUri)
        }
    });
  },

  SocUri : function () {

    if(SocUri.length>0){
        console.log('request service | Soc Uri Method | return: '+SocUri);
    }else {
        console.log('zero super nodes')
    }
   return SocUri;
  },

  resetSocUri: function () {
    SocUri = "";
  }

};
```

Sockets are used in communication. Socket.IO is a bidirectional and low-latency communication protocol. Most of dApps uses Socket.IO for communication. In sockets, we can basically listen to a particular event or trigger (emit) a event. When creating an event, it's call it .emit() while .on() is used to listen to events.

Snippet from *SocketWrapper.js*

```
function socketClientWrapper(ip) {
    console.log("Orb Service | socketClientWrapper | IP :"+ip.toString());
    IO = io(ip, { reconnect: true})
```

```
IO.once('connect', function(){

    console.log('Connected' );

    IO.on('register', function (data) {
        IO.emit('reg-req', { clientid: clientid })
        console.log('Registered' );
    });

    IO.on('event', function(data){
        console.log(data)
    });

    IO.on('new-fun', function (data) {
        let sentObj = {
            destinationSocketId: data.destinationSocketId,
            destinationClientId: data.destinationClientId,
            id: data.id,
            name: data.name,
            fn: data.fn
        }

        let fnsList = _.unionBy([sentObj], funArr, 'id')
        funArr = fnsList
        console.log('Function added to function list' );
    })

    IO.on('answer-seek', function (data, callback) {
        console.log('============ Answering machine ============')
        const params = data.params
        const fnObj = data.fnobj
        // console.log("Function array "+funArr);
        for(let fn of funArr) {
            if(fn.name == fnObj.name) {
                let actualfn = fn.fn
                let res = eval("("+actualfn+")")
                let ans = res(params)
                callback(ans)
            }
        }
        console.log('====== Answering machine ended=========')

    })

    IO.on('fnlist', function (data) {
```

```
            console.log(data)
        })

        IO.on('disconnect', function(){
            console.log('Disconneted')
        });

        IO.on('heartbeat', function (data) {
            console.log(data)
        })
    });
}
```

When deploying a new function in the system, that function is mapped into a JSON as follows. That way every nodes get updated with available functions in the network.

```
IO.on('new-fun', function (data) {
            let sentObj = {
                destinationSocketId: data.destinationSocketId,
                destinationClientId: data.destinationClientId,
                id: data.id,
                name: data.name,
                fn: data.fn
            }

            let fnsList = _.unionBy([sentObj], funArr, 'id')
            funArr = fnsList
            console.log('Function added to function list' );
    })
```

When user is trying to consume a function, the execution happens on another node in the system. So this piece of function should be extracted and run with parameters what user passes. Eval (a npm library) is used to evaluate the function and run with parameters as below.

```
        IO.on('answer-seek', function (data, callback) {
            console.log('============ Answering machine ============')
            const params = data.params
            const fnObj = data.fnobj
            // console.log("Function array "+funArr);
            for(let fn of funArr) {
                if(fn.name == fnObj.name) {
                    let actualfn = fn.fn
                    let res = eval("("+actualfn+")")
                    let ans = res(params)
                    callback(ans)
                }
            }
            console.log('====== Answering machine ended=========')

        })
```

Frontend part implementation is done using HTML, Css and Javascript. Electron has tools to navigate between pages in a way URi is hidden. So user get an experience of an app. Code snippet to map the navigation is as follows.

```
electron.config(function($routeProvider) {
    $routeProvider

        .when('/deploy', {
            templateUrl : 'deployed.html',
            controller : 'EditorController'
        })

        .when('/editor', {
            templateUrl : 'editor.html'
        })

        .when('/function', {
            templateUrl : 'callFunction.html'
        })

        .when('/wdash', {
            templateUrl : 'wallet_dashboard.html'
        })
```

```
        .when('/fstore', {
            templateUrl : 'fstore.html'
        })

        .when('/', {
            templateUrl : 'editor.html'
        })

        .when('/register', {
            templateUrl : 'register.html'
        })

        .when('/login', {
            templateUrl : 'login.html'
        })

});
```

`codeflask` (https://www.npmjs.com/package/codeflask) is an npm library used to retrieve the code inside the code editor. `node-persist` (https://www.npmjs.com/package/node-persist) npm library is used to persist data locally within the node. `javascript-obfuscator` (https://www.npmjs.com/package/obfuscator) is used to obfuscate the code before deploying. This is for code protection.

```
angular.module('orb').controller('EditorController',function
($scope,SocketService,PaymentService,PersistanceService,Web3Service) {
    //import Js file
    //  require('../js/editor')
    var CodeFlask = require('codeflask');
    const storage = require('node-persist');
    const swal = require('sweetalert');
    var JavaScriptObfuscator = require('javascript-obfuscator');
    var Interpreter = require('js-interpreter');
    var evaljs = require("evaljs");
    const perf = require('execution-time')();

    var pbar1 = $('#progressBX');
    // var pbar2 = $('#progressAA');
```

```
    pbar1.hide();
    // pbar2.hide();

//code for editor

    const flask = new CodeFlask('#code', { language: 'js' ,lineNumbers: true});

    flask.onUpdate(function (code) {
        $('#code textarea').val(code);
    });

    const code = flask.getCode();
```

Code obfuscation is a usefull technique when it comes to code protection. Code obfuscation uses several methods to protect the code. Variables renaming is the most popular use of obfuscating. Apart from that it does strings extractions and encryption on them. Dead code injection is done to complex the code. It adds junk codes inbetween and makes it complex to understand by anyone who reads to code. Control flow flattening is another technique this library includes. For an example, a code with If statements will be replaced by switch statements. Below code snippet explains the usage of obfuscator within the system.

```
// code for obfuscation

    var obfuscate = $('#obfuscate');
    obfuscate.click(function () {
        var obfuscationResult = JavaScriptObfuscator.obfuscate(
            flask.getCode(),
            {
                compact: false,
                controlFlowFlattening: true
            }
        );

        if(obfuscationResult!=""){
            console.log("Obfuscated Code : "
,obfuscationResult.getObfuscatedCode());
            swal("Obfuscated Code!", obfuscationResult.getObfuscatedCode());
```

```
    }
    else{
        toastr.warning("Please write the function first",{msg:'warning'});
    }
});
```

Function calling is a critical functionality in the system. EvalJS is used to evaluate the function and get response. Execution time of the interpreter is calculated to determine the code complexity.

```
var run = $('#runcode');
run.click(function() {
    var func = flask.getCode();

    var res = evaljs.evaluate(func)
    toastr.success('Code is Running !! ', "");

    //at beginning of your code
    perf.start();
    var myInterpreter = new Interpreter(func);
    //at end of your code
    const results = perf.stop();
    console.log("execution time "+ results.time +" ms");  // in milliseconds

    myInterpreter.run();
    console.log("Interpreter :" +myInterpreter.value);
    console.log("Evaljs :" +res.toString());

    toastr.success('Executed! Answer is :  '+myInterpreter.value.toString() ,  );
    toastr.success('Execution Time : ' + results.time ,  );
});
```

3.2.2 Node

Node is the end client of the system. Node can deploy & seed functions. Nodes are connected to bootstrap on startup. Node should acknowledge the system on leaving

and joining. Each node can vary on system specifications because its different personal devices or computers.

Node implementation contain sockets to connect, register in the network, listen to new functions, seek answers or delegate function execution to another node.

```
const clientid = ""
var socket = require('socket.io-client')('http://127.0.0.1:3300', { reconnect: true});
const express = require('express')
const app = express()
const _ = require('lodash')
const port = 3002

let funArr = []

socket.on('connect', function(){
   console.log('Connected')
});

socket.on('register', function (data) {
   console.log(data)
   socket.emit('reg-req', { clientid: clientid })

})

socket.on('event', function(data){
   console.log(data)
});

socket.on('new-fun', function (data) {
   let sentObj = {
      destinationSocketId: data.destinationSocketId,
      destinationClientId: data.destinationClientId,
      id: data.id,
      name: data.name,
      fn: data.fn
   }

   let fnsList = _.unionBy([sentObj], funArr, 'id')
   funArr = fnsList
})
```

```
socket.on('answer-seek', function (data, callback) {
    console.log('============ Answering machine ============')
    const params = data.params
    const fnObj = data.fnobj

    for(let fn of funArr) {
        if(fn.name == fnObj.name) {
            let actualfn = fn.fn
            let res = eval("("+actualfn+")")
            let ans = res(params.p1, params.p2)
            callback(ans)
        }
    }
    console.log('============ Answering machine ended============')

})

socket.on('fnlist', function (data) {
    console.log(data)
})

socket.on('disconnect', function(){
    console.log('Disconneted')
});

socket.on('answer', function (data) {
    console.log(data)
})

app.post('/', (req, res) => {
    console.log('posted.....')

    const myfun = function (a,b) {
        return a+b
    }

    const fnObj = {
        clientId: clientid,
        id: '123',
        name: 'add',
        fn: myfun.toString()
    }

    socket.emit('deploy', fnObj)
    res.status(201).send()
})
```

```
app.get('/call', (req, res) => {
   socket.emit('fcall', {clientId: clientid ,fname: 'add', params: { p1: 3, p2: 100}})
   res.status(200).send()
})

app.get('/', (req, res) => {
   res.status(200).send()
})

app.get('/:id', (req, res) => {
   res.status(200).send()
})

app.listen(port, e => {
   console.log(`Client app running at port ${port}`)
})

exports = socket;
```

Node maintains some endpoints as above to check status and live probs.

3.2.3 Supernode

Supernodes or trackers are main pillars on the network. All nodes should connect to one peer which is on lowest latency. Supernodes are capable of tracking nodes connected, maintaining their joining and leaving requests, keep track of DHTs etc.

```
var express = require('express'),
   bodyParser = require('body-parser'),
   port = process.env.PORT || 3300,
   app = express(),
   server = require('http').Server(app),
   io = require('socket.io')(server),
   p2pserver = require('socket.io-p2p-server').Server;

io.use(p2pserver);

var nodes=[];
var sock =null;

io.on('connection',function (socket) {
```

```
    sock = socket;
    console.log('Stream Build \n ID : '+socket.id);

    var arr={count:'0',socket:socket.id,port:'3100'};

    nodes.push(arr);

    socket.on('peer',function (data) {

        socket.emit('monitor', { data:{nodes:nodes}});
        console.log("peer namespace call");
    });

    socket.on('get-function',function (data) {
        console.log('server Call');
        console.log(data.length);
        if(data){
            socket.emit('returnfunction', "this is cool , function request by \n peer :
"+data);
        }else{
            console.log('empty result');
        }
    });

    socket.on('disconnect',function (data) {

        console.log('close: '+ socket.id);
    });

    socket.broadcast.emit('returnfunction', "this is cool , fu");

    socket.emit('monitor', { data:{nodes:nodes}});

});



app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

app.use('/public', express.static(__dirname +'/public'));


app.get('/',function(req,res){
    res.sendFile(__dirname +'/public/index.html');
});
```

```
server.listen(port,"0.0.0.0",function(){
   console.log("Supernode Initiated:"+port);
});
```

Supernode also maintains endpoints to check status.

3.2.4 Smart contact

Smart contract is used to automate the fund transactions in a decentralized app. Its a stored procedure that has predefined way of sharing funds in the network.

Solidity is used to write the smart contract. Solidity is the most popular programming language for smart contracts. Its object oriented and high level programing language. (It supports inheritance).

```
pragma solidity ^0.8.4;

contract OrbContract {
    address public minter;
   mapping (address => uint) public balances;

   event Sent(address from, address to, uint amount);

   constructor() {
      minter = msg.sender;
   }

   function mint(address receiver, uint amount) public {
      require(msg.sender == minter);
      balances[receiver] += amount;
   }

   error InsufficientBalance(uint requested, uint available);

   // Sends an amount of existing coins
   // from any caller to an address
   function send(address receiver, uint amount) public {
      if (amount > balances[msg.sender])
         revert InsufficientBalance({
            requested: amount,
            available: balances[msg.sender]
```

```
        });

        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

The line event Sent(address from, address to, uint amount); declares an “event”, which is emitted in the last line of the function send. Ethereum clients such as web applications can listen for these events emitted on the blockchain without much cost. As soon as it is emitted, the listener receives the arguments from, to and amount, which makes it possible to track transactions.

To listen for this event, you could use the following JavaScript code, which uses web3.js to create the Coin contract object, and any user interface calls the automatically generated balances function from above:

```
Coin.Sent().watch({}, '', function(error, result) {
    if (!error) {
        console.log("Coin transfer: " + result.args.amount +
            " coins were sent from " + result.args.from +
            " to " + result.args.to + ".");
        console.log("Balances now:\n" +
            "Sender: " + Coin.balances.call(result.args.from) +
            "Receiver: " + Coin.balances.call(result.args.to));
    }})
```

The constructor is a special function that is executed during the creation of the contract and cannot be called afterwards. In this case, it permanently stores the address of the person creating the contract. The msg variable (together with tx and block) is a special global variable that contains properties which allow access to the blockchain. msg.sender is always the address where the current (external) function call came from.

The functions that make up the contract, and that users and contracts can call are mint and send.

The mint function sends an amount of newly created coins to another address. The require function call defines conditions that reverts all changes if not met. In this example, require(msg.sender == minter); ensures that only the creator of the contract can call mint. In general, the creator can mint as many tokens as they like, but at some point, this will lead to a phenomenon called “overflow”. Note that because of the default Checked arithmetic, the transaction would revert if the expression balances[receiver] += amount; overflows, i.e., when balances[receiver] + amount in arbitrary precision arithmetic is larger than the maximum value of uint (2**256 - 1). This is also true for the statement balances[receiver] += amount; in the function send.

Errors allow you to provide more information to the caller about why a condition or operation failed. Errors are used together with the revert statement. The revert statement unconditionally aborts and reverts all changes similar to the require function, but it also allows you to provide the name of an error and additional data which will be supplied to the caller (and eventually to the front-end application or block explorer) so that a failure can more easily be debugged or reacted upon.

The send function can be used by anyone (who already has some of these coins) to send coins to anyone else. If the sender does not have enough coins to send, the if condition evaluates to true. As a result, the revert will cause the operation to fail while providing the sender with error details using the InsufficientBalance error.

If you use this contract to send coins to an address, you will not see anything when you look at that address on a blockchain explorer, because the record that you sent coins and the changed balances are only stored in the data storage of this particular

coin contract. By using events, you can create a “blockchain explorer” that tracks transactions and balances of your new coin, but you have to inspect the coin contract address and not the addresses of the coin owners.

CHAPTER 4

RESULTS & EVALUATION

4.1 Overview of the chapter

This chapter is focussed on general functionality of the system, results, testing & evaluation. General functionality include all the steps from installation to function deployment and function calling. Reason to include this under this chapter is because this show the end results to the user. This shows how the system looks for an external user. This chaper will also discuss things like testing based on performance, accuracy, security and functionality.

4.2 General functionality

When the client starts the "Orb" client software, it will send a request to the Supernode and the Supernode will start to keep track of the particular node. When the user uploads a role through the client, the client will ask the user to subscribe to a payment plan (an Ethereum smart contract). Then the function will run on the same client machine first. Then the source code of the function will be run through a hash function and a hash will be applied.

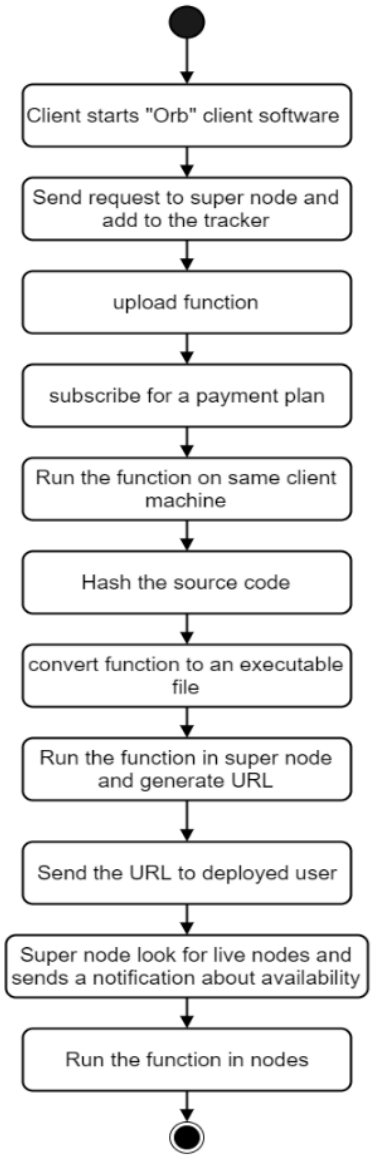


Figure 5.3 Process of function deployment

4.2.1 Deployment & distribution of functions in the decentralized network

This will generate an initial hash that can be used to identify the function. Also, a key pair will be generated for a particular function. Then the function will become an executable file. Then the function will be sent to the Supernode. The Supernode will initially execute the function, generate a URL, and send it to the implementer. Then the Supernode will search the list of live nodes and send a notification about the

function availability. Nodes can now choose to run the function or not. In this way, the functions will be distributed throughout the network.

4.2.1.1 Function deployment steps

As pre-requisites for deployment, we need to setup the Node and install "Orb" client on it. This "Orb" client will be downloadable from web an executable file which can be installed easily on any personal computer or device. This client is to be developed in the headless mode too.

4.2.1.1.1 Download & Setup Orb

This application is developed as a cross platform application. This can be run on any operating system available right now. For example,

- Microsoft Windows (x86, x86_64 and arm64 architecture)
    - .EXE executable file
    - Traditional MSI installer
    - AppX package for the Windows Store
    - Squirrel (Windows-based installer - from the Electron maintainers group)
- Linux (x86, x86_64, arm64, armv7l and mips64el architectures)
    - .DEB executable file
    - .RPM executable file
    - Flatpak file
    - Snap file
- MacOS (x86_64, arm64, and universal architectures)
    - .DMG executable file

After downloading the relevant executable, Install the Client and start the application.

On the start, You will see a dashboard with analytic data on the user’s activity. Things on this page can be configured as desired. For example, We can configure it to show the Ether balance on the main page. This displays the Ether balance in the wallet connected to this client.

On the left side, There is a navigation menu which user can use to switch between the windows. Navigate menu contains options like Dashboard, My functions, Available functions window, Inline code editor window, Function calling window, Wallet related views and FAQ.

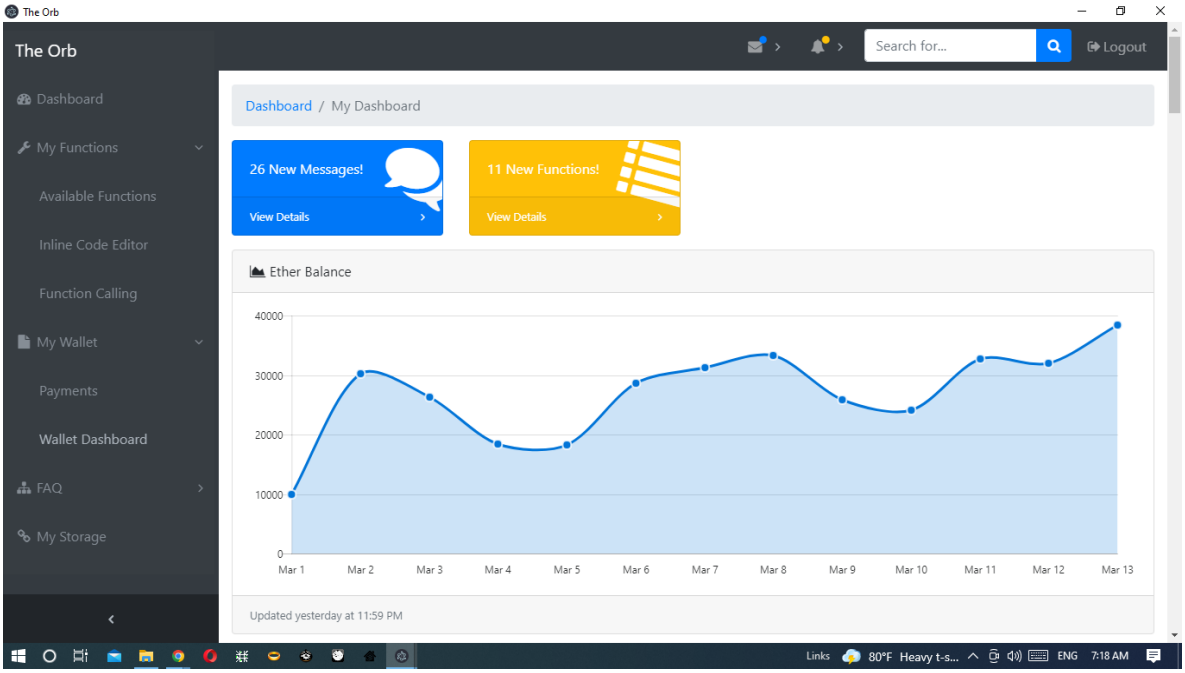


Figure 5.4 "Orb" Dashboard

4.2.1.1.2 Implement code on Inline code editor

Inline code editor is used to write, test and deploy a function code. User can open this windows, Write a suitable function name and start coding. Function ID is auto generated at deployment. This is used to identify the function uniquely in the system. This is a lengthy code with more than 16 characters.

There is a convention on writing the piece of function code. The function needs to accept arguments as a single record and output single record. Function should give transparency by logging errors and exceptions where possible.

All function should be single responsible. For example, User can write a function that analyses a sentence. But that function shouldn't analyze images. User can keep

separate functions on different concerns. But it's not acceptable to keep all inside one function.

The function code should be readeable. Functions are sharing among the users/devolopers. So knowledge of the users/devolopers differs from one to one. Thats why code should be in a way that its readeable to anyone who reads through it. In future "Orb" will design a system where quality of the function can be estimated in the system. This can be done before deployment.

"Orb" editor currently can facilitate only code editing, running and deployment. Features like code formating, debugging, auto suggesting features are yet to come.

Conventions that should be maintained in the function code

- Function should log everything.
    - Errors
    - Exceptions
- Function should return at the end.
- Function should be single responsible.
    - Shouldn’t do many things in a single function.
- Function should be readeable.
    - Add code comments where possible.
    - Add doc comments.
- Code should be formatted.
    - Inline code editor will improve to facilitate more functions like this.

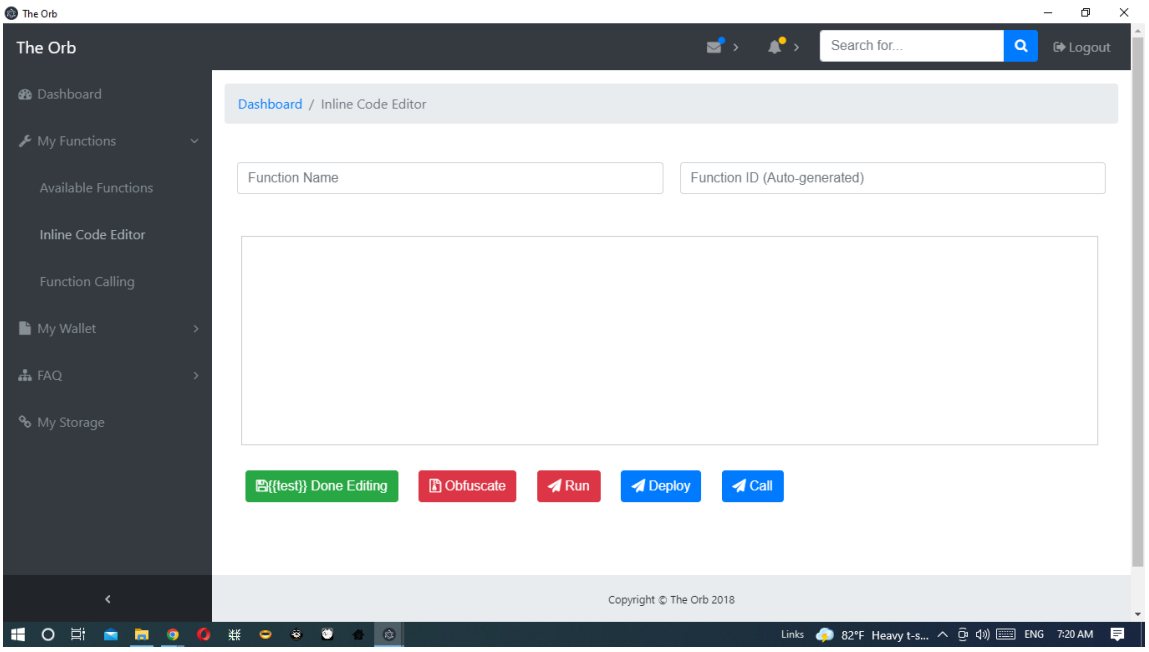


Figure 5.5 "Orb" Inline code editor

Example code

Here is a sample code that is going to be deployed on Orb. This function is written in Nodejs. This can be used for simple addition.

```
function add(param) {
    let total = 0
    for (let elem in param) {
        total += param[elem];
    }
    return total;
}
```

Table 5.1 Simple "Add" function

The code will be displayed as below.

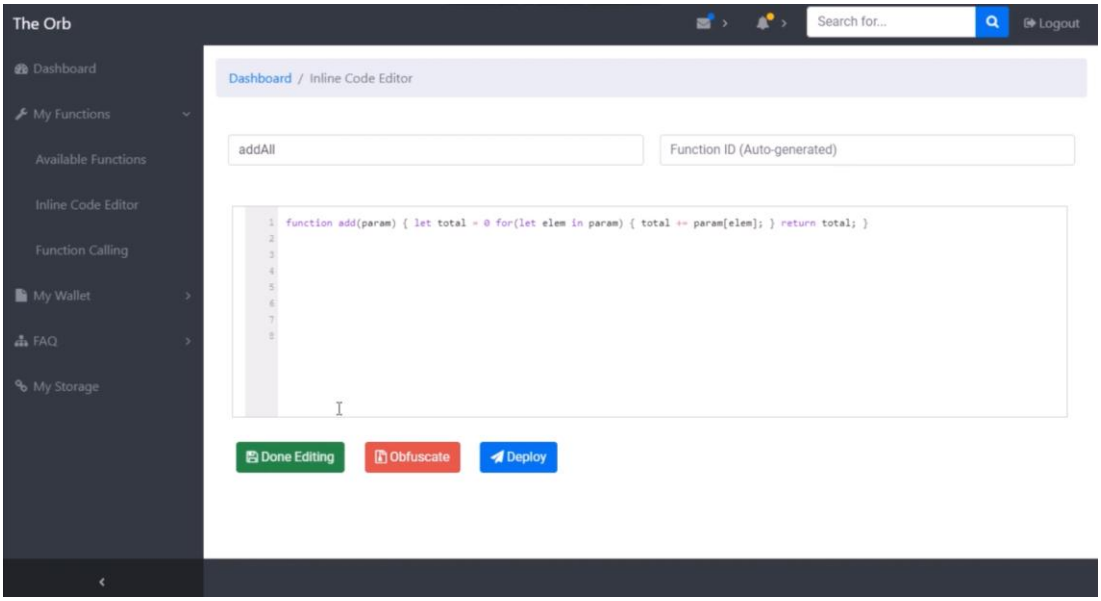


Figure 5.6 "Orb" Add function in inline code editor

4.2.1.1.3 Code obfuscation

Once the function is complete, We can deploy it. Before deployment, function should be obfuscated for security purposes. Users are not encouraged to have sensitive data on the function code anyways.

Code obfuscation is a process that converting code to a state what is not useful to hackers, but functional. This process changes the method metadata, instrunctions but it doesn't change the final output of the code. This way it assure the functionality.

Platforms like Xamarin, C #, VB.NET, F#, Java, Android, iOS, & .NET, consists of free decompilers that can be activated at any time and installed without any effort. The source code can be returned from the library. An automated code debugging program makes reverse engineering difficult and economically impossible.

There is a button on the code editor to obfuscate the code and what it does is renaming methods names, variables, string encryption, etc.

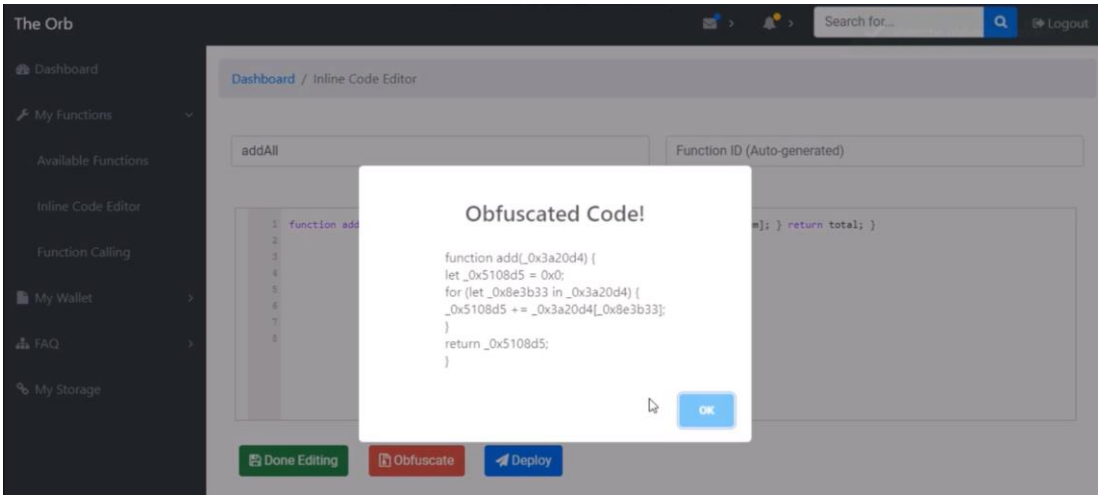


Figure 5.7 "Orb" Obfuctation code

Several methodologies are used in obfuscating the code here.

Names in the code will be in a variety of suffixes, such as "a", "b", "c", or numbers, non-printable letters or invisible letters. This is called "Rename obfuscation" in computer terminology.

Original Source Code Before Rename Obfuscation

```
private void
CalculatePayroll (SpecialList employee-
Group) {
   while (employeeGroup.HasMore()) {
        employee =
employeeGroup.GetNext(true) ;
        employee.UpdateSalary() ;
        Distribute Check(employee) ;
    }
}
```

Reverse-Engineered Source Code After Rename Obfuscation

```
private void a(a b) {
   while (b.a()) {
        a = b.a(true);
        a.a () ;
        a.(a) ;
    }
}
```

Figure 5.8 Rename Obfuscation

In a managed and activated one, all the fibers are clearly detectable and readable. Even when renaming methods and variables, threads can be used to locate critical code segments by searching for fiber references in binary.

This includes messages that are displayed to the user (especially error messages). In order to provide an effective barrier against these types of attacks, fiber encryption hides activable fibers and restores their original value only when necessary. Fiber decoding during runtime usually carries a short runtime performance fine. This is called “String encryption” and this is also a methodology used in obfuscation.

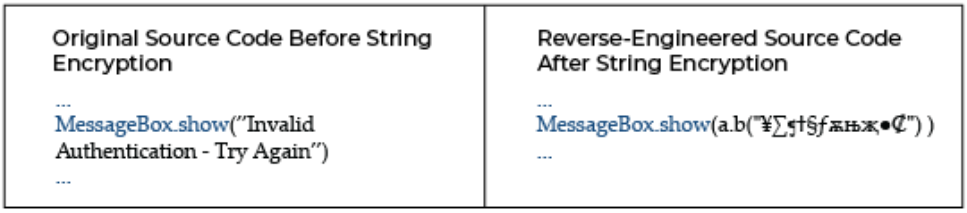


Figure 5.9 String Encryption

Control flow blurring synthesizes conditional, branching and iterative constructions that produce a valid actionable argument, but yields undefined interpretive results when disassembled. This methodology is called “Control flow obfuscation” in this regard.

Simply put, the decomposed code looks like a spaghetti argument and is very difficult for a hacker to understand. This technique will affect the running time of a system.

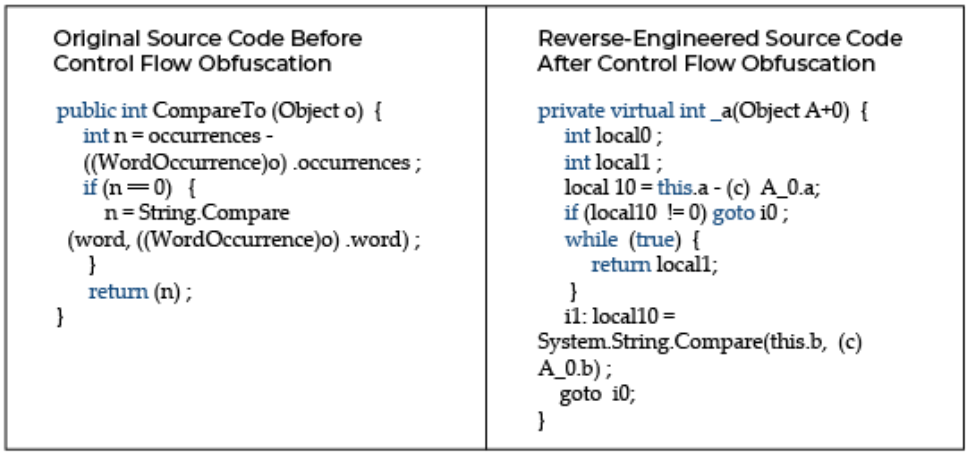


Figure 5.10 Contro-flow Obfuscation

4.2.1.1.3 Deploy to the network

Final step is deploying. On deployment, users need to pay a gas price. This is used for the initial deployment and it is paid in Ether. User can pay from his wallet and deloy the function.

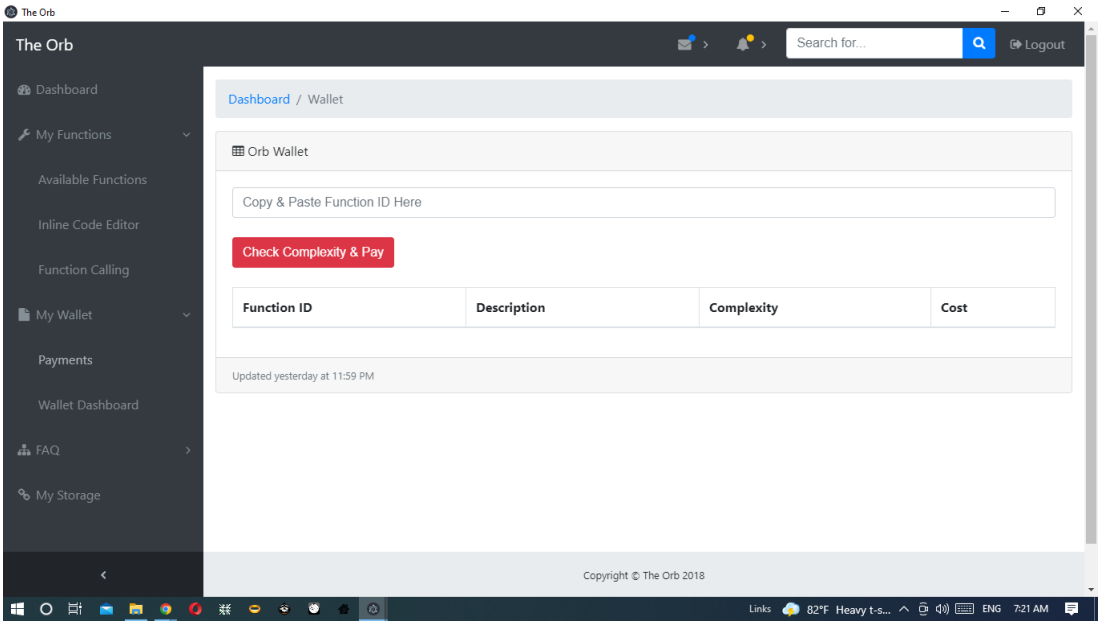


Figure 5.11 "Orb" Deploy to network

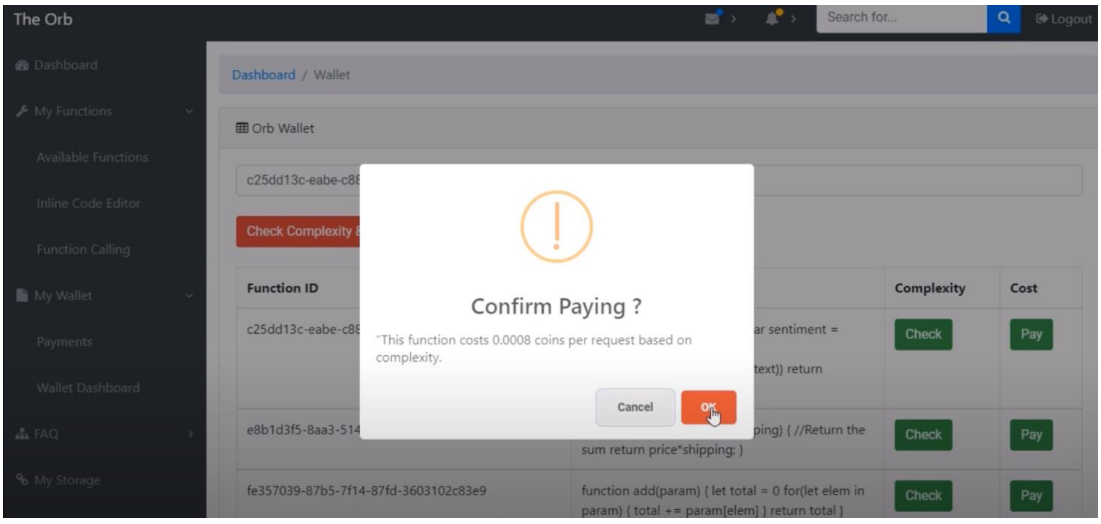


Figure 5.12 "Orb" Confirmation of payment

Amount of the gas price depends on the complexity of the function. More complex functions needs more gas value while less complex functions need less gas value. At the same time, Function owner earns more out of a complex function compared to a simple function.

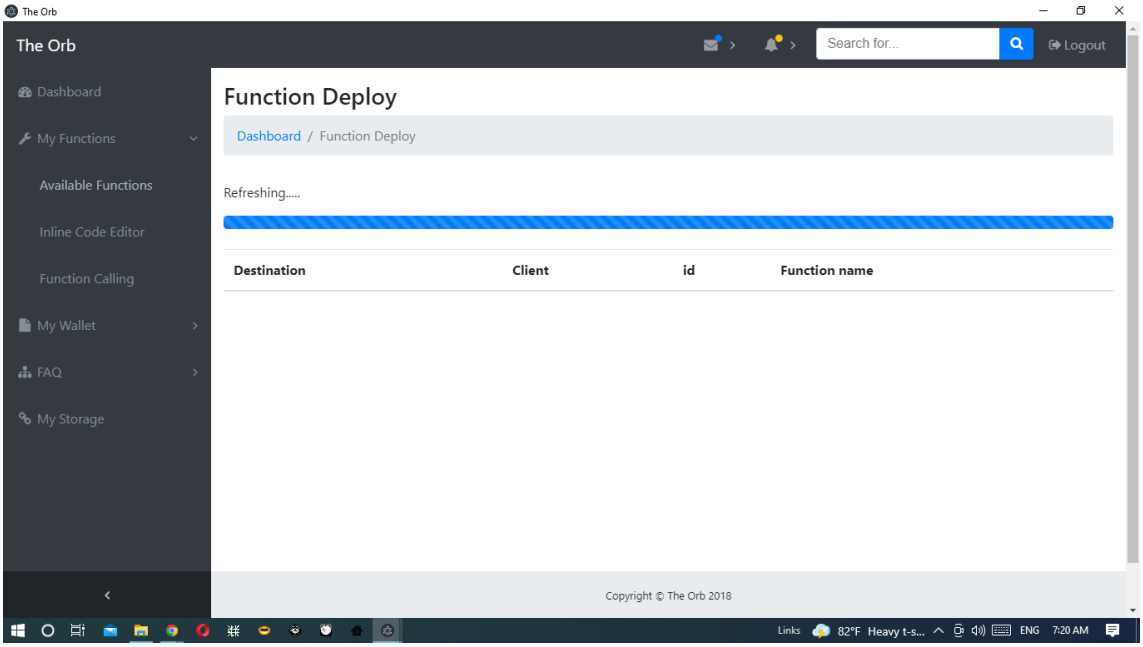


Figure 5.13 "Orb" Function loading window

Once the function is deployed, User will be redirected to the above page. This page consists of all the functions available in the network.

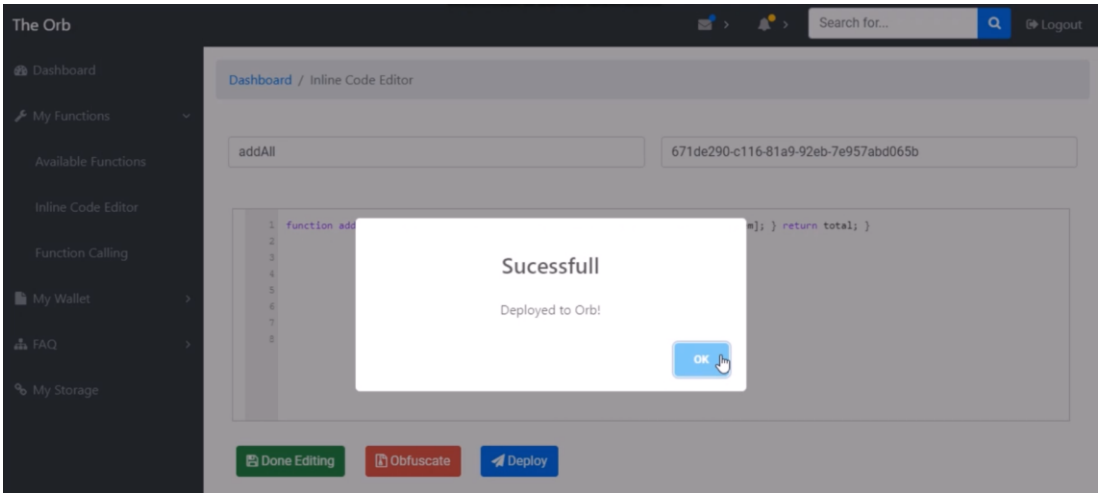


Figure 5.14 "Orb" Successful deployment

4.2.1.2 Function distribution/propogation

Function distribution or propagation in the network happens automatically in a periodical & timely manner. Once a function is deployed, First its deployed to the most nearest Supernode. Then it's propagating between Supernodes in a timely fashion. It takes considerable amount of time but still its subjective depending on the size the network.

"Orb" is using a peer-to-peer lookup protocol called Chord[31] or efficient deployment. This makes the system more scalable because all functions are not located near all Supernodes in this protocol.

Basic and primary theory on this Chord protocol is selecting a efficient location for a data item. In Chord[31], nodes adapts efficiently to join and leave the system. Location of the data [32] is determined by analyzing key/data pair. Chord[32] periodically does consistent hashing to assign keys to nodes.

Because all nodes in the chord receives approximately the similar no of keys and requires relatively small number of keystrokes when the nodes are connected to and out of the system, the hashing tends to be balanced continuously.

In other protocols of consistent hashing assumes each node is aware of many other nodes in the system. It does not scale well to a large number of nodes. Conversely, every chord[33] node requires "routing" information. Hence the system cannot scale because of that.

This way one node needs to keep only few routing information about other nodes. Hence lookup table is small and speed up the communication within the system.each Chord[34] node needs "routing" information about only. This way it maintains distributed routing tables for efficient communication.

In a stable state, in an N-node system, each node maintains information only about O (log N) other nodes, and resolves all look-ups via O (log N) messages to other nodes.

4.2.2 Calling a deployed function

Function can be called once deployed. User who deployed the function can test that function by recalling as an end user who uses that function to get something done.

User needs to click on function calling tab on the left side of the user interface to get into the screen. Then user loads the list of functions available to execute by clicking dropdown “function name”.

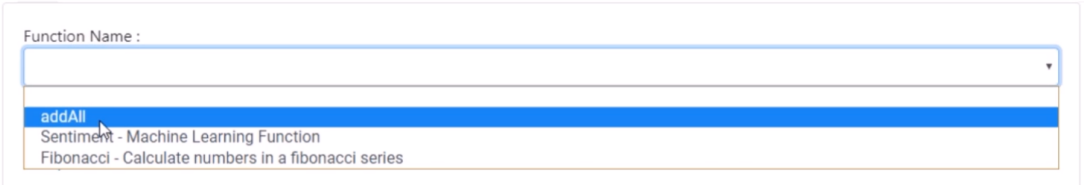


Figure 5.15 “Orb” List functions

Once the function is selected, User can input the parameters needed for that function in the following text box, which is right below that (User can input argument as JSON too. Click on that text “Accept JSON as a parameter”. This will pop up a window)

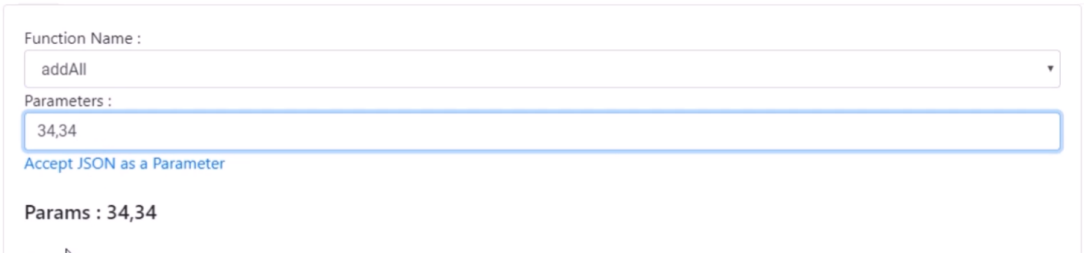


Figure 5.16 “Orb” Add values to parameters

Now press the “Call” button to execute.

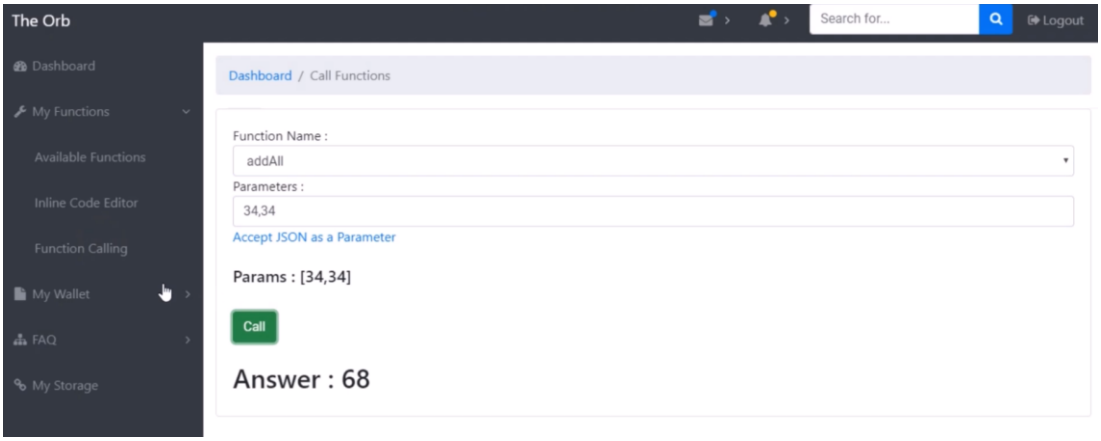


Figure 5.17 "Orb" Executing funtion

This will call the respective function as gives back the expected output. If you check the logs of the system, you can check what node has involved in this computation to provide back the results.

4.2.3 Updating and depreciating of a function

The deployed user can initiate a deprecated function or an update. So this request will be propagated through the network. The update and deactivation process will be slow as the system is decentralized.

4.2.4 Handling payments through wallet

"Orb" is capable of handling payments through an attached wallet. User can connect his/her wallet to this client application and perform payments from that. The following screenshot displays the UI designed for the wallet.

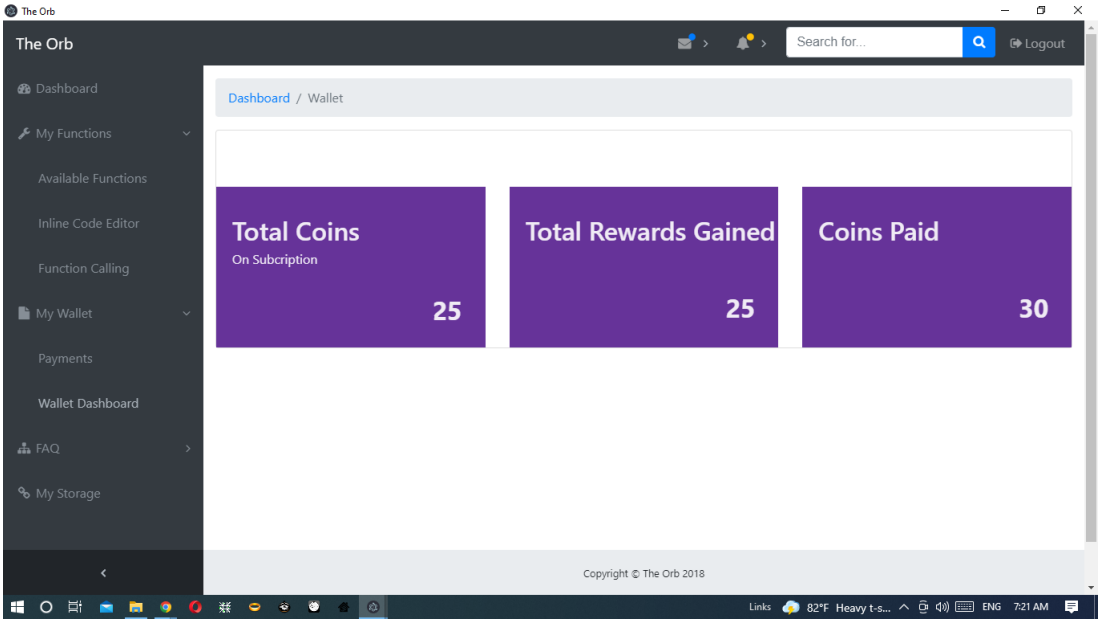


Figure 5.18 "Orb" Wallet dashboard

This user interface displays,

- Total coins remaining
    - Amount of Ether available to spend.
- Total rewards gained
    - Rewards gained out of function invocations.
- Coins paid
    - Ether used to invoke functions or deploy functions (gas price)

Here the pricing model is pretty simple and understandable. Network doesn't take any amount from the transactions. This is totally transparent to the user and user is full confident of using the system.

Every user gets 50 coins on registration to the system. This is as provided as coins for trail and error. User can try the system as a newbie, use those coins to deploy and call functions. User needs to import Ether to its account after initial free 50 coins.

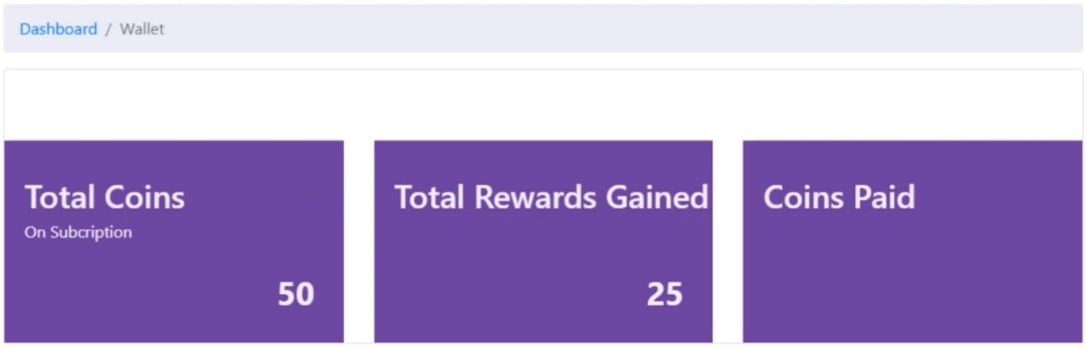


Figure 5.19 "Orb" Wallet

4.2 Samples scenarios

4.2.1 Global temperature warning mechanism using IoT

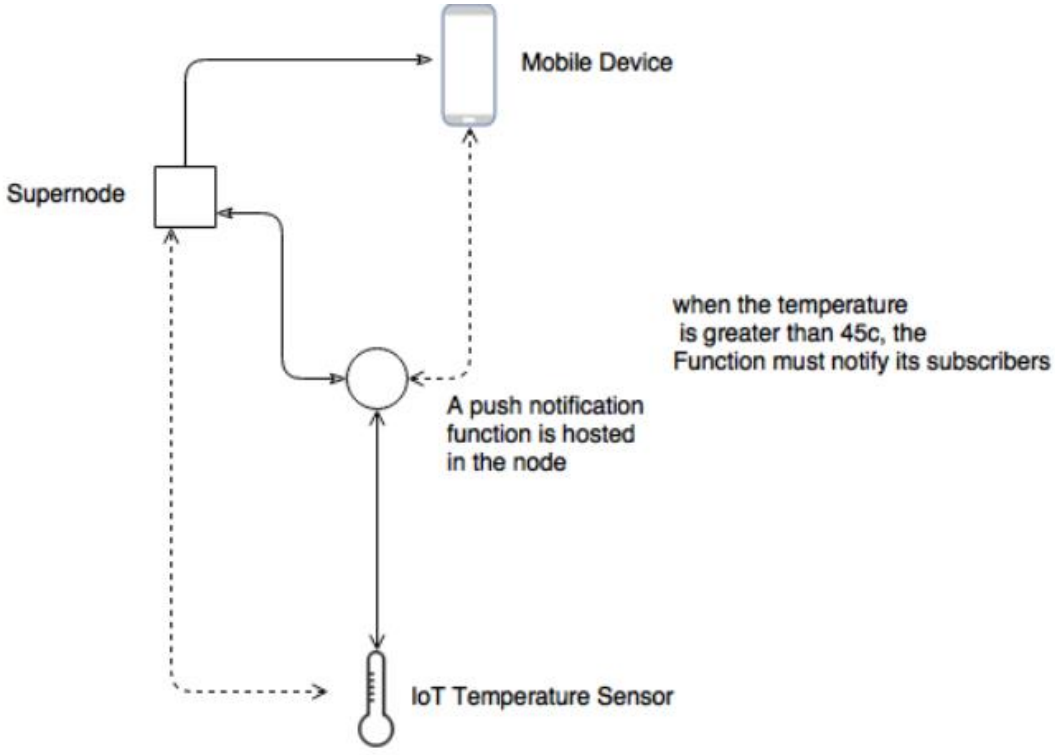


Figure 5.20 Global temperature warning mechanism using IoT

Imagine there are thousands of Supernodes and 10,000 clients contributing to the Orb. If a node connects, it will first subscribe to the closest Supernode and create a secure

connection immediately after deployment. The node will receive a list of other Supernodes and nodes to which it can connect. In this way, a single node creates a peer-to-peer connection. When a user implements a function, using the process mentioned above, the function will be copied and starts running on the different nodes. According to the diagram above, a push notification function is running on different nodes and there are mobile clients subscribed to the push notification function and consumes the push notification function. The IoT temperature sensor initially connects to a Supernode, gets a list of pairs, and sets p2p connections with the other nodes that execute the push notification function. Now, the IoT function will send real-time data at 5-minute intervals to the nodes, and the function determines whether to send notifications to subscribers or not. When there is a temperature peak, the IoT sensor will call the function of the pair with the lowest latency. So it will run and all subscriber mobile devices will be notified. Since the push notification functions are decentralized and run on 10,000 nodes, there will be limited traffic. Since the function runs on many nodes, availability will be very high even if nodes are offline. The system will tolerate any number of IoT devices as Supernodes decide where to connect. With the ethereum-based payment method, users which host the push notification feature will be paid for. For the owner to reduce the cost of hosting, the user can host functions.

4.2.2 Sharing a reusable function code

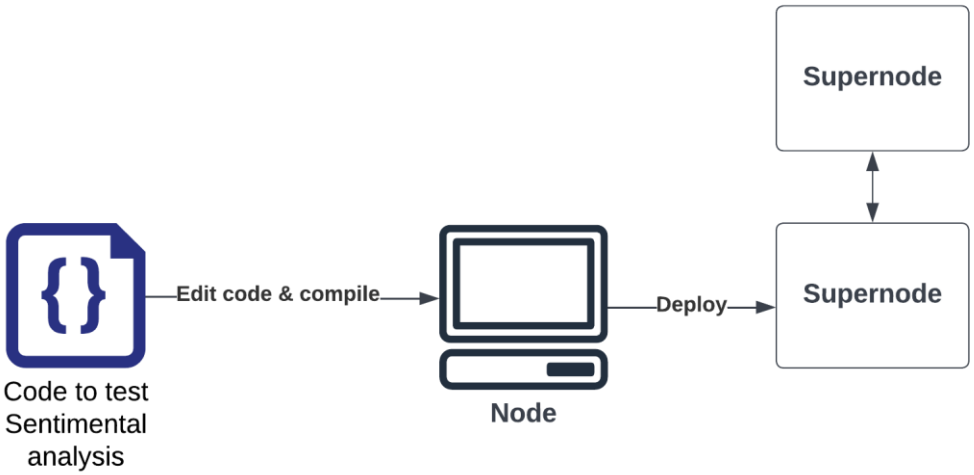


Figure 5.21 Sharing a reusable function code

First user needs to write a code that can be sharable in the network. That piece of code shouldn’t contain personal credentials, sensitive data, etc. At the same time this code needs to be meaningful and usable for another user in the network. For example, Sentimental function will get a statement (text) as an input and returns whether that statement is positive, negative or neutral, depending on the words it has used. The Sample code for the above scenario is as follows,

```
function sentiment(param) {
        var sentiment = require("node-sentiment")
        console.log(sentiment(param.text))
        return sentiment(param.text)
}
```

Table 5.2 Sentimental function

User needs to write this function with proper parameters on inputs. These parameters need to be documented to increase code readability. User can use the “"Orb" code editor” to edit & run code. If it works fine, User can upload it to "Orb" network. User needs to pay some amount of Ether on function deployment (as Gas).

Once this is successfully deployed from the "Orb" application, it replicates among the Supernodes making it available for the whole network. This successfully deployed function is visible for other nodes to use after some time. This delay depends on the size of the network.

If some user wants to use that function, User needs to subscribe to that function and use it. User will be charged per usage only. Usage is calculated per service call. Here node is basically seeding that particular function when using. So when there are many users requesting that function, the seed count becomes high.

When requesting (or using) a deployed function, Requester will be charged and that amount will be directly owned by the function owner. Since this is a decentralized system, No central authority owns the credit in middle. This way function owners earn more.

4.3 Testing

Testing is an essencial step in the process of delivering a quality product to the end-user. Testing benchmarks ensures desired quality of a product. “Orb” is designed on top of a p2p system. So non functional requirements like global scalability, host availability, trust, anonymity and deniability should be considered on testing.

Functional requirements are related to functionality or features provided by the “Orb”. The main functional requirement of “Orb” is to facilitate user to deploy a function on the decentralized network in a way that can used by other nodes. There are many other functional requirement built around this.

4.3.1 Non functional testing

The demand for internet based applications are increasing day by day. So any system should be capable of scaling to facilitate the growing demand. The goal of “Orb” is to achieve global scalability which enables decentralized function as service model to be accessible from anywhere in the world.

Global scalability is bounded with some constraints like network bandwidth, no of nodes, supernodes, spread of nodes and supernodes, capacity and performance of nodes etc. Good balance of these constraints lead to a globally scalable, high performance, fault tolerant system.

“Orb” should possess a right combination between no of nodes & supernodes. If millions of nodes are managed by few supernodes, System performance fails drastically. Supernodes must be distributed across the network overlay and it should not be clustered within subregions of the overlay.

Nodes must have lower latency when accessing closest supernode in the system. High latency values negatively affect the system accessibility. Supernode to supernode communication must also be run with lower latencies despite network and bandwidth delays.

Supernodes play the role of load balancing in a p2p system. So supernodes should identify the traffic and should be capable of handling growing demands. One supernode have a capability limit on serving nodes. So there is a K value which represents the maximum number of nodes that can be connected to a supernode at a time. This K value differs according to the performance and capacity of the supernode.

Peer-to-peer system has heterogeneity among nodes and supernodes. Different nodes have different performance and capacities. Node needs a defined minimum level of hardware specification to connect to the network.

Nodes should be able to install the “Orb” application. To install “Orb”, device needs an opering system with minimum 1 GB of RAM and 10GB free storage. CPU architecture can be x86, x86-64 or AMD64. This is about minimum requirements. 4GB of RAM and 15GB free storage is recommended for a good performance.

Peer-to-peer system are having a dynamic behavour. Nodes & supernodes leaves and join the system frequently. But system should be adaptable to handle this leaving and joining requests seamlessly. Specially supernodes should be resistant from this node churn.

Supernodes plays an important role in security. Malicious supernodes can spread false information in the system and disrupt the total system. Apart from that supernodes can be subjected to denial of service (DDoS) attacks.

4.3.1 Functional testing

Functional testing validates the features of “Orb” client and residing network. So the main function of deploying a function code should be tested first. Then other functional requirements like user registration, code obfuscation, code editing, payment handling, calling functions also should be tested.

As the first step of testing these features are tested at the time of development in unit testing. In unit testing individual components are tested with expected output against the actual output.

After initial development of the application, Alpha & beta testers tests the application for potential bugs. This goes in a cycle for some time until major bugs are fixed and the beta release is confident to be released as general available product (GA) .

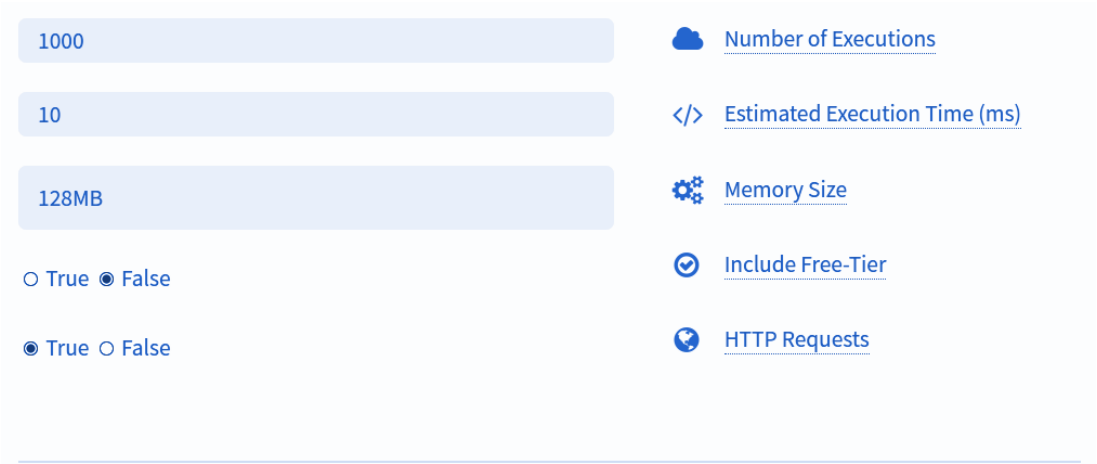
After GA, End users starts to keep of using the features of the system. This is a kind of black box testing since end user have less/no idea on what is happening inside. After getting the feedbacks from few end users, “Orb” will be releasing patches on top of GA to fix the bugs identifed.

Feedback forms, detecting user activity on “Orb” client are the ways of collecting information about the overall experience of “Orb” user.

4.4 Evaluation

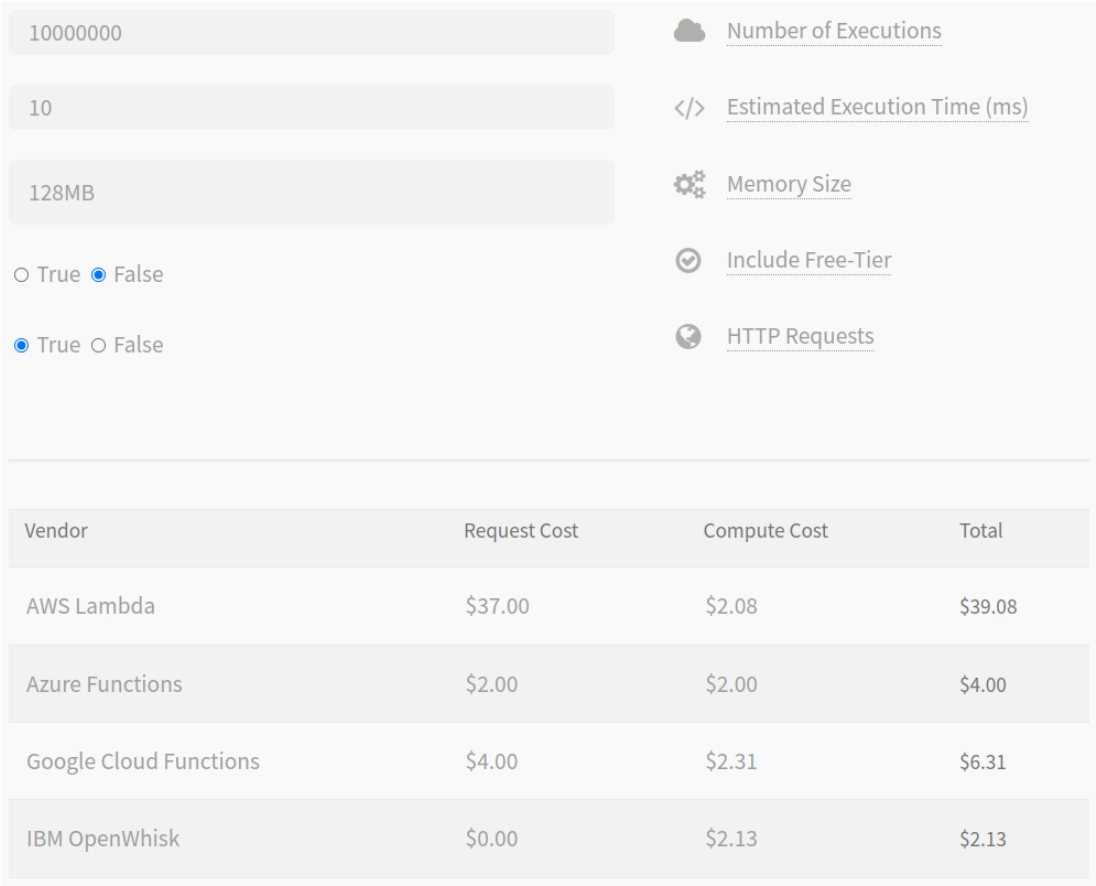
4.4.1 Cost evaluation

Most FaaS providers use pay-as-you-go & pay-per-request payment models which is cost effective to the user. To calculate estimated cost, every cloud providers have their own costing tool. There are some open-source tools that are used to compare costing with cloud providers. Serverlesscalc [46] is a one of that kind.



Vendor	Request Cost	Compute Cost	Total
AWS Lambda	$0.00	$0.00	$0.00
Azure Functions	$0.00	$0.00	$0.00
Google Cloud Functions	$0.00	$0.00	$0.00
IBM OpenWhisk	$0.00	$0.00	$0.00

Here you can see there is no charge for first 1000 executions on any network. But after 1000 there is a charge and it differs from vendor to vendor. Here is the estimated cost for 10000000 requests in leading FaaS providers.



Vendor	Request Cost	Compute Cost	Total
AWS Lambda	$37.00	$2.08	$39.08
Azure Functions	$2.00	$2.00	$4.00
Google Cloud Functions	$4.00	$2.31	$6.31
IBM OpenWhisk	$0.00	$2.13	$2.13

Speciality in “Orb” is you can earn from the system while you are paying for the system. The above centralized vendors doesn't have a feature like that. The below table show the pricing per request comparing with all major FaaS providers.

OBJ

4.4.2 Performance evaluation

There are no limits on performance of the “Orb” since load balancing is handled by the network. But centralized FaaS providers have limits on memory, execution time, payload size etc.

	MEMORY (MB)		EXECUTION TIME		PAYLOADS (MB)	
	Default	**Max**	**Default**	**Max**	**Request**	**Response**
Non-Edge offerings						
Vercel Functions	1024	3008	10s-900s[2]	10s-900s	5	5
Netlify Functions	1024	1024[3]	10s[3]	10s[3]	6	6
IBM Cloud Functions	256	2048	1m	10m	5	5
Google cloud Functions	128	2048	NA	540s	10	10
Oracle Functions [4]	128	1024	30s	120s	6	6
Azure Functions [5]	NA	1536	5m	10m	No limit	No limit
AWS Lambda	128	3008	3s	15m	6[1]	6[1]

Note : Azure have no limits in payload size. But all others have limits.

4.5 Research findings

As findings of this research, modified p2p network infrastructure that facilitates public/private network communication and a rewarding model that runs on Etherium blockchain-powered by smart contracts can be taken mainly.

The main motivation for this research was current drawbacks and issues on centralized function as a service infrastructure. The initial idea was to decentralize the centralized function as a service with all other similarities there. But later on, the evolution of decentralized web 3.0 and dApps that rewards the users with coins improved this idea to make it a model that rewards the user. That motivated this Etherium-based smart contract.

During the observation of the initial MVP, It showed that there is a need for UI to elaborate the system's functionality to the user. That motivated me to implement a

fully functional client studio with the ability to search for functions in the network, deploy functions, consume functions, maintain a wallet, etc.

As per the cost evaluations above, You can see most FaaS providers don't charge oAs findings of this research, modified p2p network infrastructure that facilitates public/private network communication and rewarding model that runs on Etherium blockchain-powered by smart contracts can be taken mainly.

The main motivation for this research was current drawbacks and issues on centralized function as a service infrastructure. The initial idea was to decentralize the centralized function as a service with all other similarities there. But later on, the evolution of decentralized web 3.0 and dApps that rewards the users with coins improved this idea to make it a model that rewards the user. That motivated this Etherium-based smart contract.

During the observation of the initial MVP, It showed that there is a need for UI to elaborate the system's functionality to the user. That motivated me to implement a fully functional client studio with the ability to search for functions in the network, deploy functions, consume functions, maintain a wallet, etc.

As per the cost evaluations above, You can see most FaaS providers don't charge for the first 1000 executions. But after that, the price increases massively. But in this solution, We found a way to reduce cost; by utilizing the freely available computation power and storage in the personal computers that are connected to this network.

This way it eliminates the cloud provider's involvement in storing user data and providing computation power. This network is a crowd-sourced, crowd-funded, self-evolving network.

As per the performance evaluations above, there is an improved performance increment in this solution compared to the traditional existing centralized function as a service. In centralized FaaS providers, there is a limit on maximum computations, latency, memory, and payload size. But in this solution, there is no limit as such. When more users are connected to the system, that means there are more nodes in the system,

and it is possible to cater to a large number of users with computation power, storage, and low latency.

CHAPTER 5

CONCLUSION & RECOMMENDATION

5.1 Overview

Goal of this chapter is to conclude all the information gathered in the research to compile a conclusion based on that. Benefits of this research, revisited objectives, limitations of this research component, target markets, marketing strategies and future work is explained in undet this chapter.

5.2 Benefits of this research

There are numerous benefits on this research. This research component “Orb” is a good replacement for existing FaaS providers. This enables the user to earn while using the system.

End users of this systems are 2 types. One is a developer who knows to code. Developers can contribute to the system by deploying functions. Other user type is non-technical user. These users can execute the function calls available in the system.

Users are not charged for system maintaince or cost of infastrucure. Users only pay for what is being used and what function consumer pays to the network will directed transferred to function owner.

5.3 Revisited objectives

The objectives of the research are revised and summarized as below. Objectives determines the system that is going to achieve at the end. At the same time objectives provided a guided approach for top down analysis.

“Orb” has 7 main objectives to be achieved at the end.

1. User must be able to deploy a service (function code) on the decentralized network.

2. Implementing distributed hash table (DHT) on supernodes that keep track of live nodes in the system.
3. Facilitating all features of a typical FaaS on a decentralized network.
4. Payment model that provides a fair reward for all users contributing on the network.
5. Smart contract for handling payments from contributors and consumers.
6. Wallet implementation that enables the user to sync own wallet rather than creating a new wallet. (Bring-your-own-wallet)
7. Implementing a tamper-proof network resistant to DDoS attacks.

5.4 Limitations

Main limitation in “Orb” is around updating deployed functions. Updating anything on a decentralized network is a tedious task because we have no single point of updating.

As a step to overcome this. “Orb” release versions on same functions instead on updates. Whenever user wants to update a function, new version of that function will be released to the network. So consumers always take the latest version on consuming which can eliminate this limitation.

5.5 Target Market

"Orb" is a network infrastructure that allows developers and non-developers to reuse functions and allow them to use basic hardware to perform the same tasks as cloud servers. As there are many types of developers and it is a broad market sector, our target market sector will be "Developers using cloud servers to run and perform serverless functions."

5.6 Marketing Strategy

Rewarding system is the main attraction point here. The ability to earn cryptocurrencies from what you are doing daily as a developer is a good marketing point. Developers usually like open-source and sharing. Here devolopers can contribute to the network and earn easily. It’s just a matter of separating out logics that

are written everyday to solve problems. So other users can use the functions without the knowledge of coding and credit the original owner of the function. This crowd sourced system can potential to grow on its own.

5.7 Contribution

- Research on existing FaaS models, compare and contrast between them.
- Model that can address current drawbacks in existing infastructure.
- Identifying aspects that needs improvements in existing infastructure.
- Research on usage of WebSockets for this model.
- Network design (evolved from P2P concepts) that handles communication between nodes, supernodes using sockets.
- Research on libraries that facilitate Socket communication and finding best out of them that meet benchmarks of the network.
- Keeping track of nodes using DHTs (Chord).
- Maintaining lowest latency in network communication.
- Address issues related to node leaving and joining.
- Implementing reliable status management in node communication.
- Initial idea of rewarding the user who contributes to the system.
- Research on best cryptocurrency (Etherium) that is suitable for this system.
- Research on Ether/Wei transactions and gas prices for each transactions.
- Research of model of charging the user on function deployment, seeding and consuming.
- Attaching a crypto-wallet to the user's account to perform transactions.
- Solidity based smart contract to define the logic behind each transaction and pre-define the share for each party involved.

5.8 Future work

"Orb" provides the functionality of basic FaaS provider on a decentralized network with some value additions like rewarding model based on contributions. This is the first implementation of a decentralized FaaS network. As initial implementation, this is a MVP that facilitates the basic functionality.

We can focuss on following areas as next steps.

- Identifying malicious supernodes using AI based algorithm.
    - Malicious supernodes can bring the whole network down by spreading false information in the network. So AI based algorithm should detect these malicious nodes and active remove them from the network.
- Version management on function deployment.
    - Editing a deployed funtion on a decentralized network is crucial. Since versions needs to be managed during edits.
- Bring-your-own-wallet that enables the user to plug his/her own crypto wallet to “Orb”
    - “Orb” is creating a new wallet on user creation as of now. But user should have the option of creating a new wallet of bringing his own wallet for this application.

REFERENCES

[1]D. Etherington, "Amazon AWS S3 outage is breaking things for a lot of websites and apps", TechCrunch, 2021. [Online]. Available: https://techcrunch.com/2017/02/28/amazon-aws-s3-outage-is-breaking-things-for-a-lot-of-websites-and-apps/. [Accessed: 16- Sep- 2021].

[2]M. Roberts, "Serverless Architectures", martinfowler.com, 2021. [Online]. Available: https://martinfowler.com/articles/serverless.html#unpacking-faas. [Accessed: 14- Sep- 2021].

[3]"What is Ether", Ethereum.org, 2021. [Online]. Available: https://www.ethereum.org/ether. [Accessed: 18- Sep- 2021].

[4]"How Do Ethereum Smart Contracts Work? - CoinDesk", CoinDesk, 2021. [Online]. Available: https://www.coindesk.com/information/ethereum-smart-contracts-work/. [Accessed: 14- Sep- 2021].

[5]"ConsenSys – Medium", Medium.com, 2021. [Online]. Available: https://medium.com/@ConsenSys. [Accessed: 21- Sep- 2021].

[6]"What Is AWS Lambda? - AWS Lambda", Docs.aws.amazon.com, 2021. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. [Accessed: 22- Sep- 2021].

[7] “Steemit,” Steemit. [Online]. Available: https://steemit.com/. [Accessed: 24-Mar-2021].

[8]“What is Steemit and how does it work?”, www.inverse.com, 2021. [Online]. Available: https://www.inverse.com/article/41699-what-is-steemit-how-does-it-work. [Accessed: 25- Mar- 2021].

[9]"AWS Rates Highest on Cloud Reliability", EnterpriseTech, 2021. [Online]. Available: https://www.enterprisetech.com/2015/01/06/aws-rates-highest-cloud-reliability/. [Accessed: 14- Mar- 2021].

[10]"The AWS Outage: The Problem with Internet Centralization | Mondo", Mondo, 2021. [Online]. Available: https://www.mondo.com/aws-outage-internet-centralization-problem/. [Accessed: 14- Sep- 2021].

[11]"Single Point of Failure – What is it? Why it Matters... :", Renovodata.com, 2021. [Online]. Available: http://www.renovodata.com/blog/2015/06/10/single-point-of-failure. [Accessed: 14- Sep- 2021].

[12]"www.ey.com", Ey.com, 2021. [Online]. Available: http://www.ey.com/Publication/vwLUAssets/EY-implementing-blockchains-and-distributed-infrastructure/$FILE/EY-implementing-blockchains-and-distributed-infrastructure.pdf. [Accessed: 14- Sep- 2021].

[13]P. Labs, "IPFS is the Distributed Web", IPFS, 2021. [Online]. Available: https://ipfs.io/. [Accessed: 14- Sep- 2021].

[14]"Storj - Decentralized Cloud Storage", Storj - Decentralized Cloud Storage, 2021. [Online]. Available: https://storj.io. [Accessed: 14- Sep- 2021].

[15]Google Cloud Platform Documentation | Documentation | Google Cloud. [Online]. Available: https://cloud.google.com/docs/. [Accessed: 02-Feb-2021].

[16]“Kubernetes Documentation,” Kubernetes. [Online]. Available: https://kubernetes.io/docs/home/?path=users&persona=app-developer&level=foundational. [Accessed: 02-Feb-2021].

[17]“Decentralized Internet on Blockchain – Hacker Noon,” Hacker Noon, 03-Oct-2021. [Online]. Available: https://hackernoon.com/decentralized-internet-on-blockchain-6b78684358a. [Accessed: 03-Oct-2021].

[18]Blockchain Based Distributed Control System for Edge Computing - IEEE Conference Publication. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7968630. [Accessed: 02-Apr-2021].

[19]“Decentralizing Your Microservices Organization,” The New Stack, 02-Apr-2021. [Online]. Available: https://thenewstack.io/decentralizing-microservices-organization/. [Accessed: 02-Apr-2021].

[20]“Building Your First Network,” Building Your First Network - hyperledger-fabricdocs master documentation. [Online]. Available: http://hyperledger-fabric.readthedocs.io/en/release-1.0/build_network.html. [Accessed: 02-Apr-2021].

[21]M. Meister, “leveraging Kubernetes to run a private production ready Ethereum network,” Medium, 02-Apr-2021. [Online]. Available: https://medium.com/@cryptoctl/leveraging-kubernetes-to-run-a-private-production-ready-ethereum-network-b6f9b49098df. [Accessed: 02-Apr-2021].

[22]A Decentralized Service Discovery Approach on Peer-to-Peer Networks - IEEE Journals & Magazine. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/5928313/. [Accessed: 02-Apr-2021].

[23]“Taming WebRTC with PeerJS: Making a Simple P2P Web Game,” Toptal Engineering Blog. [Online]. Available: https://www.toptal.com/webrtc/taming-webrtc-with-peerjs. [Accessed: 02-Apr-2018].

[24] “Decentralized payments for environmental services: The cases of Pimampiro and PROFAFOR in Ecuador,” Ecological Economics, 31-Dec-2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921800907005320. [Accessed: 05-Jan-2022].

[25] A. Crespo and H. Garcia-Molina, “Semantic Overlay Networks for P2P Systems,” SpringerLink, 19-Jul-2004. [Online]. Available: https://link.springer.com/chapter/10.1007/11574781_1. [Accessed: 02-Mar-2018].

[26] “Rethink the Web browser,” Beaker | Peer-to-peer Web browser. No blockchain required. [Online]. Available: https://beakerbrowser.com/. [Accessed: 05-Feb-2021].

[27]"Statista - The Statistics Portal for Market Data, Market Research and Market Studies", Statista.com, 2021. [Online]. Available: https://www.statista.com/. [Accessed: 15- May- 2021].

[28]"How Do Ethereum Smart Contracts Work? - CoinDesk", CoinDesk, 2018.[Online]. Available: https://www.coindesk.com/information/ethereum-smart-contracts-work/. [Accessed: 15- May- 2021].

[29] “Serverless by numbers“, 2018. [Online]. Available: https://serverless.com/blog/serverless-by-the-numbers-2018-data-report/. [Accessed:05- October- 2021].

[30] Mcanini.github.io, 2022. [Online]. Available: https://mcanini.github.io/papers/ez-segway.sosr17.pdf. [Accessed: 02- April- 2022].

[31] "Chord: Building a DHT (Distributed Hash Table) in Golang", Medium, 2022. [Online]. Available: https://medium.com/techlog/chord-building-a-dht-distributed-hash-table-in-golang-67c3ce17417b. [Accessed: 05- April- 2022].

[32] G. Kecskemeti, Applying integration techniques and methods in distributed systems and technologies.

[33] Kth.se, 2022. [Online]. Available: https://www.kth.se/social/upload/51647996f276545db53654c0/3-chord.pdf. [Accessed: 07- April- 2022].

[34]"Code Obfuscation: A Comprehensive Guide Against Reverse-Engineering Attempts - AppSealing", AppSealing, 2022. [Online]. Available: https://www.appsealing.com/code-obfuscation/. [Accessed: 07- April- 2022].

[35] Resources.mpi-inf.mpg.de, 2022. [Online]. Available: https://resources.mpi-inf.mpg.de/d5/teaching/ws03_04/p2p-data/11-18-writeup1.pdf. [Accessed: 07- April- 2022].

[36] "What is P2P(Peer-to-peer process) ? - GeeksforGeeks", GeeksforGeeks, 2022. [Online]. Available: https://www.geeksforgeeks.org/what-is-p2ppeer-to-peer-process/. [Accessed: 07- April- 2022].

[37] "Peer to Peer Network | Purposes, Methods, Types & Facts", Teach Computer Science, 2022. [Online]. Available: https://teachcomputerscience.com/peer-to-peer-network/. [Accessed: 07- April- 2022].

[38]"How (and Why) to Obfuscate Source Code | Embroker", Embroker, 2022. [Online]. Available: https://www.embroker.com/blog/how-to-obfuscate-source-code/. [Accessed: 07- April- 2022].

[39]"Peer to Peer (P2P) Network? Architecture, Types, and Examples", DigitalThinkerHelp, 2022. [Online]. Available: https://digitalthinkerhelp.com/what-is-peer-to-peer-p2p-network-with-architecture-types-examples/. [Accessed: 07- April- 2022].

[40] "What Is a Peer-to-Peer (P2P) Network? | Indeed.com", Indeed Career Guide, 2022. [Online]. Available: https://www.indeed.com/career-advice/career-development/what-is-a-peer-to-peer-network. [Accessed: 07- April- 2022].

[41] J. Cope, "What's a Peer-to-Peer (P2P) Network?", Computerworld, 2022. [Online]. Available: https://www.computerworld.com/article/2588287/networking-peer-to-peer-network.html. [Accessed: 08- April- 2022].

[42]"What is Code Obfuscation? | Guardsquare", Guardsquare.com, 2022. [Online]. Available: https://www.guardsquare.com/what-is-code-obfuscation. [Accessed: 08- April- 2022].

[43] Cloud.ibm.com, 2022. [Online]. Available: https://cloud.ibm.com/docs/openwhisk?topic=openwhisk-getting-started. [Accessed: 01- May- 2022].

[44] "Apache OpenWhisk is a serverless, open source cloud platform", Openwhisk.apache.org, 2022. [Online]. Available: https://openwhisk.apache.org/. [Accessed: 01- May- 2022].

[45] "IBM Cloud Docs", Cloud.ibm.com, 2022. [Online]. Available: https://cloud.ibm.com/docs/solution-tutorials?topic=solution-tutorials-serverless-api-webapp. [Accessed: 01- May- 2022].

[46]"Serverless Cost Calculator", Serverlesscalc.com, 2022. [Online]. Available: http://serverlesscalc.com/. [Accessed: 16- May- 2022].

[47] Docs.soliditylang.org. 2022. Solidity — Solidity 0.8.15 documentation. [online] Available at: <https://docs.soliditylang.org/en/v0.8.15/> [Accessed 25 June 2022].