Fall 2021

Achieving Fairness through Load-Balancing in Social Cloud Computing Networks

Kaiyi Huang
San Jose State University

Achieving Fairness through Load-Balancing in Social Cloud Computing Networks

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Kaiyi Huang

Nov 2021

The Designated Thesis Committee Approves the Thesis Titled

Achieving Fairness through Load-Balancing in Social Cloud Computing Networks

By

Kaiyi Huang

Approved for the Department of Computer Science

San Jose State University

Dec 2021

Dr. Navrati Saxena	Department of Computer Science
Dr. Ben Reed	Department of Computer Science
Dr. Abhishek Roy	Senior Cloud Engineer (MediaTek Inc.)

Abstract

Cloud-based computing networks have taken over the digital landscape. From small non-profits to large multinational corporations, more and more entities have been offloading computing effort to the cloud in order to take advantage of the increased cost-efficiency and scalability of cloud computing. One of the new types of cloud that have emerged is the P2P cloud, which disengages from a traditional datacenter setup by allowing users to instead share their own computing hardware into a cloud to take advantage of cloud computing's advantages at an even lower cost. However, this new paradigm comes with a slew of challenges, notably, security when operating with the devices of strangers and fairness when not operating in a market-based system. This paper aims to address these two issues by proposing an algorithm based on social credits for a P2P cloud system that uses a social network to establish its security measures. The project implements the Social Credit algorithm along with two other task-migration based load balancing algorithms adapted for a P2P social cloud in Cloudsim Plus, and the results are compared in terms of general metrics and fairness.

Acknowledgements

I would like to greatly thank my advisor, committee members and my parents for following along with me throughout this journey. The project took a delay due to real life issues along the way and I deeply appreciate them for putting up with the issues.

I would like to thank Dr. Navrati Saxena for supervising my research from the start and setting me on the right direction to complete it. Without her, I probably couldn't have come up with enough material to make this truly new research.

I would like to thank Dr. Ben Reed for supporting me throughout the process and being lenient with my time issues.

I would like to thank Dr. Abhishek Roy for taking the time out from his busy schedule to help comment and guide me on my research. The relevant cloud metrics and analysis wouldn't have been possible without him.

I would also like to thank Dr. Manoel Campos da Silva Filho for maintaining the CloudSim Plus project and working with me in a timely manner to fix blocking bugs that was preventing the experimentation portion from working.

And lastly, I would like to thank my parents for supporting me and giving me the opportunities needed to pursue academia while working full-time.

Tables of Contents

List of Figures

List of Tables

I. Introduction

Cloud computing has become an indispensable aspect of our digital landscape [14]. With the advent of cloud computing, businesses and organizations have been able to migrate infrastructure and workload online, making more efficient use of resources and at a lower cost and effort to the users in question. Cloud computing also brings with it many new challenges to computing, between the risks of offloading critical business logic and sensitive information onto third-party hardware, to the headaches of managing thousands and even millions of users who are sharing computational hardware.

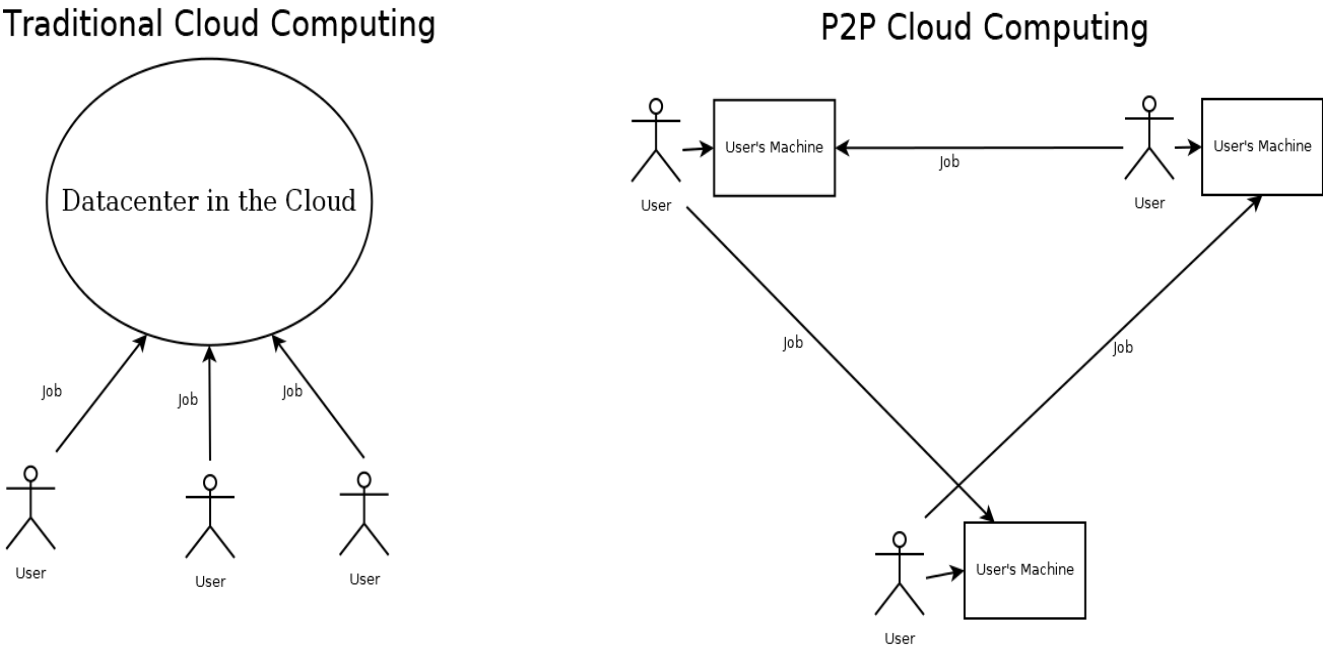

Figure 1 - Comparison between Traditional and P2P Cloud

A recent development is the emergence of peer-to-peer cloud computing [6, 12], where instead of a defined separation of end-users and datacenters, the end-users contribute to the computational power of the cloud and serve as the datacenter with their own devices. P2P clouds, especially BYOD(Bring you own device) clouds have much greater flexibility and even

lower costs compared to a conventional cloud computing setup due to the way the hardware is utilized [12] [15]. However, a peer-to-peer cloud computing network offers even more challenges compared to a traditional one, as processing on end users' devices is far less secure than a traditional datacenter and the nature of users as both consumers and producers of jobs means that a credit system needs to exist instead of a traditional pricing structure.

One of the solutions to this end is using the existing mechanisms from a social network to secure the cloud in terms of individual users. A resulting social cloud [2] can have users pick out "friends" that they trust and assign security values to their jobs, with the highest security tasks restricted to friends only while lower security tasks can be farmed out to the friends of the friends, or the friends of friends of friends, and so on. As analyzing the individual security of each node in a user-based P2P cloud computing network is difficult without invasive access, letting users decide their own security levels and their trusted partners is a viable method of implementing security for a P2P cloud.

Another challenge that a P2P cloud faces is fairness [7], which is more of an issue with it than for traditional cloud computing where users are not job consumers. Simply put, a user in a P2P cloud network should be getting rewarded (in terms of shorter processing times for their jobs and less stress on their own machine) in return for the price they are paying (processing other people's jobs). This fairness can be resolved via a market mechanism [6], but in a more cooperative scenario, a different approach can be used, which is the purpose of this paper.

II. Objectives

The purpose of this research project is two-fold. Firstly, the project aims to explore the implications of a social cloud network regarding load-balancing and VM migration. The restrictions imposed by a social network mechanism on the busy and resource-limited P2P network raises several concerns, the most significant of which is the issue that truly optimal VM allocation is not possible in a social cloud. As the complex web of user-to-user friend relationships are an additional imposed rule on the VM allocation mechanisms which traditionally cared not for which machine the job is executed on as long as it is within the SLA. This presents a difficult situation for the traditional load-balancing and allocation mechanisms that the newly hypothesized algorithm will be compared to. At the most basic level, it will examine how the respective algorithms can cope with the necessity of utilizing suboptimal configurations.

Secondly, this project aims to optimize fairness for a P2P cloud network. While there have been other research on fairness in a P2P standard [7], there hasn't been one focusing on it on the user level. Fairness is a concept not often utilized in traditional cloud paradigms and is resolved in a capitalist manner in existing research [6]. This project aims to seek a more cooperative manner of fairness stemming from the ability of a social cloud network. Metrics that the allocation algorithms tested to this end are:

- Price: Power Expended in Processing Cloudlets/Amount of cloudlets sent: Effectively, the user should pay a price for each job length sent to the cloud to be processed by others in terms of CPU power consumed. This number should be as consistent as possible between the users to be fair, but its magnitude doesn't really matter.

- Reward: Total Cloudlet time of a user's jobs/Cloudlets processed: The user should be rewarded with a decreased amount of time for each cloudlet they themselves process. This value should be desired as low as possible across all users for efficiency measures, but for fairness all it matters is that it's consistent between users.

- Total processing time: The general measure of efficiency in terms of CPU time, increasing fairness should not increase this value significantly or else we're sacrificing performance for fairness, which is a no-go.

- Total Cloudlet Time (Actual time): The sum of how long each cloudlet spent in the system, including how much time was spent queuing. This is beyond the CPU time measured in the previous metric but is more relevant as a statistic from the perspective of a user in the P2P cloud.

- Amount of overloaded hosts: The amount of overloaded hosts is a general indicator of how efficient a semi-predictive allocation algorithm is. Minimizing this value is a positive.

- Total amount of VM migrations: Effectively the cost in extra bandwidth in excess of what the cloudlets demand baseline. Minimizing this value is desirable to conserve bandwidth costs and indirectly reduce processing time.

III. Related Work

Chard et al.'s papers, directly about the social cloud, [2] [3] both lay out the basics of a social cloud. A Social cloud network is based off of existing social network paradigms, which Chard's paper posits as an application built based off of Facebook's existing infrastructure. Chard's paper goes into detail with the parts it needs, beginning with the application portion of the network, the virtualized resources needed to support the network, and the banking, registration and service marketplaces that the cloud needs to support a market-based framework where computing services are offered for sale by the users of the social cloud. This implementation does not bother with load-balancing, as it expects the users themselves to work out what their individual SLAs and workloads are.

Chang et al.'s paper about social private clouds [4] offers a less individualistic take on social clouds, and establishes trust and its counterpart, risk, as key aspects of a P2P network that a social cloud can handle with proficiency. Resource trading is also established as an important part as users must contribute and consume each other, with a marketplace that can be either market-driven or centrally-regulated (the latter of which is used in this paper). More importantly, the paper also lays out the concept of social capital [3], which directly inspired the social credit system which can be considered to be a concrete theory and implementation of the concept to an actual system. A reputation-based algorithm is further elaborated on in Wu's paper about a Peer-to-Peer reputation mechanism [18] mathematically, and the algorithm introduced in this paper borrows somewhat from the Probabilistic Estimation Measure introduced.

Mohaisen et al. [5] Adds in some considerations on how an attacker would target a social cloud, and implements their own version of a trust-based algorithm for task scheduling. This is however

treating the relative trust between the hosts in a social cloud as a recommendation value to be considered for migrations rather than as an ironclad rule that untrusted machines for a user cannot be used to run irs jobs. Its implementation of trust however does inspire certain design patterns in this paper's social credit system, which functions off of a similar basis.

Babaoglu et al's paper [6] as well as Falcão et al.[7], Joe-Wong et al. [8] and Tang et al. [9] establishes some of the parameters for fairness in the context of a cloud computing network, but not necessarily a P2P one. This paper establishes some parameters for fairness which were taken as a basis for the fairness metrics discussed within this paper. Fairness is laid out in general as how even the service providers treat each individual customer and how even the service is provided to them in terms of time, computing capacity, and other metrics. Much emphasis is also made on the trade-offs in efficiency and revenue when improving fairness, which is reflected in the experimental results of this paper.

Tajamolian et al. [10], Ibrahim et al. [11], Li et al. [12], and Alam et al. [13] detailed different methods of performing migration-based load balancing. While Ibrahim's paper was the algorithm that was chosen (other than one of the simple naïve implementations) to serve as a point of comparison to the algorithm in this paper, all of the papers served to both inform some thoughts behind the implementation of the algorithm through their varied implementation approaches.

Li et al. focuses on the use of UAVs in a cloud based setting as the nodes of a system in need of task migration. The base algorithm that they are modifying is the Virtual Machine Migration Algorithm Based on Threshold (VMBTA) which is among the most popular virtual machine methods used today to reduce system load and increase efficiency across the system.

The algorithm actually introduced is a rather straightforward addition to VMTBA that only runs VMTBA on a probability and performs a migration only for non-data intensive tasks if the probability is lower than the dice roll. The experiment results showed that this procedure significantly improved results when there is a lot of tasks being executed with the processes having to compete for resources, showing that conventional algorithms might require refinement and that altering the environment is another form of parameter for the experimental results. This paper was mostly relevant for how it detailed its Cloudsim implementation.

CloudSim [1] is used as the simulation platform for this project, and appears to be a generically good option for simulations in general, as it runs on Java which is largely platform independent, and does not rely on any hardware factors to analyze the results.

[16] The trust and reliability paper ultimately became the core of my planned algorithm. It explores an environment very similar to a P2P cloud in terms of stability and introduces a trust value based upon the unreliability of the machine, which forms the basis of the active migration portion of the algorithm introduced in this paper. The active migration portions is what ultimately gives the proposed algorithm teeth and lets it increase fairness greatly compared to a generic load balancing algorithm. The calculations for that trust value also influenced the calculation of social credit for this paper.

IV. Load Balancing Algorithms

The most straightforward algorithms are those that simply aim to initially place and migrate VMs to whatever host that can fit them, according to a general pattern. RR(Round Robin) and Best Fit algorithms are both examples of this type of algorithm. As described in [17], Round Robin load balancing distributes VMs evenly around the hosts, one after the other, both on startup and at run-time to try and achieve an even balance. The RR load balancing technique can be used both dynamically and statically [17], and when used dynamically it will also attempt to migrate VMs from overloaded hosts to available hosts in a round-robin manner as well, without regard to how appropriate a host is for the VM as long as it is possible to fit the VM within the host without overloading.

A Best Fit allocation policy, on the other hand, does account for the host's current state when migrating. To fit in the maximum amount of VMs per host, it chooses the host from the list of available hosts with nothing migrating in and can fit the VM, and picks the host that is currently *most* occupied in terms of the processing units used by VMs. This host is considered the Best Fit host and will receive the VM to be migrated out of an overloaded host. This algorithm is more appropriate for a social cloud as the limited amount of recipients that a migrating VM can possibly go to given the security restrictions means that fitting more VMs onto hosts should be a priority.

Ultimately, both of these algorithms are relatively naïve approaches that do not perform anything more complex than a **O(n)** scan of the list of appropriate hosts for a VM to make a decision. In

the experiment the Best Fit algorithm is used to compare to the proposed algorithm, but the Round Robin algorithm achieved very similar result to the Best Fit algorithm in the metrics used.

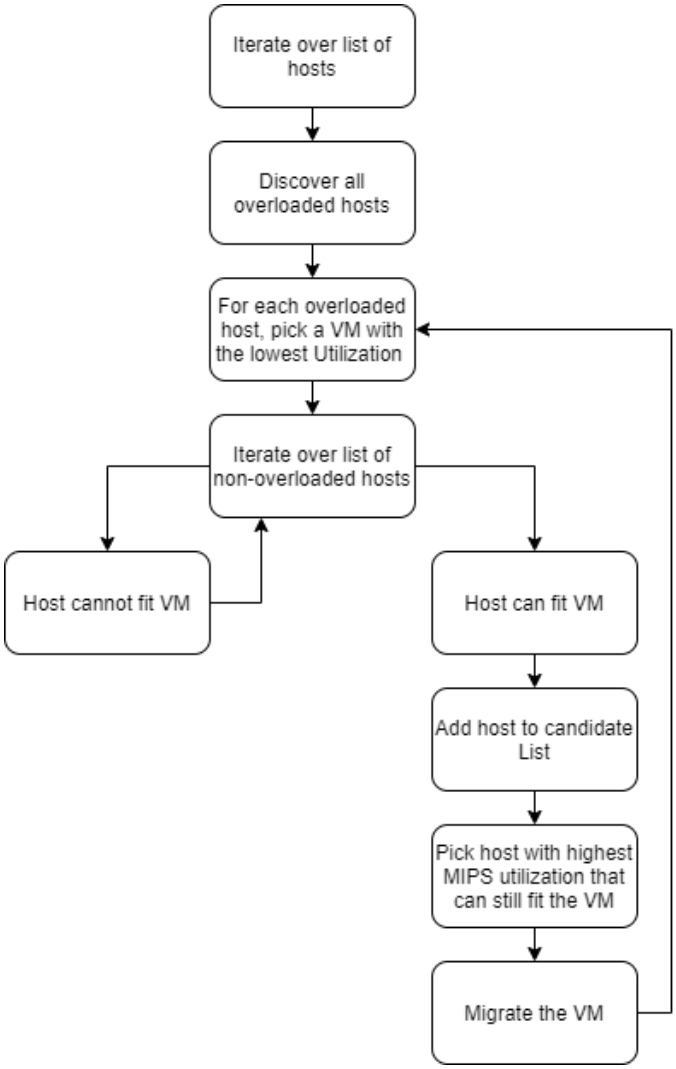

Figure 2 - Best Fit Load Balancing Migration Algorithm

EAMA [11] is a more complex algorithm used primarily with the goal of reducing migration times and reducing the total amount of migrations. EAMA works in four phases. The first phase detects overloaded hosts via a simple algorithm that scans the list of hosts and turns the list over to the migration phase of the algorithm. The second phase, underloaded host detection, is handled in a similar way and also turns the list over to the migration algorithm. The migration algorithm takes these lists and functions as thus:

1. Sort the underloaded hosts in descending order by their migration delay in relation to a designated nearby location. For the purposes of our social cloud which does not have a centralized location, this is instead replaced by the number of friend hosts of the underloaded host to achieve a similar result
2. Iterate over the list of overloaded hosts
3. Sort the overloaded hosts' VMs in ascending order by the amount of MIPS (processing power) currently being utilized by the VM.
4. If the underloaded host at the head of the list is capable of receiving the VM at the head of their list, migrate the VM to the underloaded host and move that underloaded host to the normal host list. If not, move on to the next underloaded host.
5. Repeat until either all underloaded hosts are gone, in which case begin migrating VMs from overloaded hosts to the normal utilization host list and end when that is done, or all overloaded hosts have been relieved, in which case move on to step 6.
6. Split the list of remaining underloaded hosts by half, and migrate the VMs from one of the halves into the other. Then power down the machines that now have no VMs on them.

This algorithm is expected to perform well in terms of general metrics due to being a mature algorithm, but not in fairness as it is not optimized for it. It may also introduce complications due to the restrictions a social cloud network imposes.

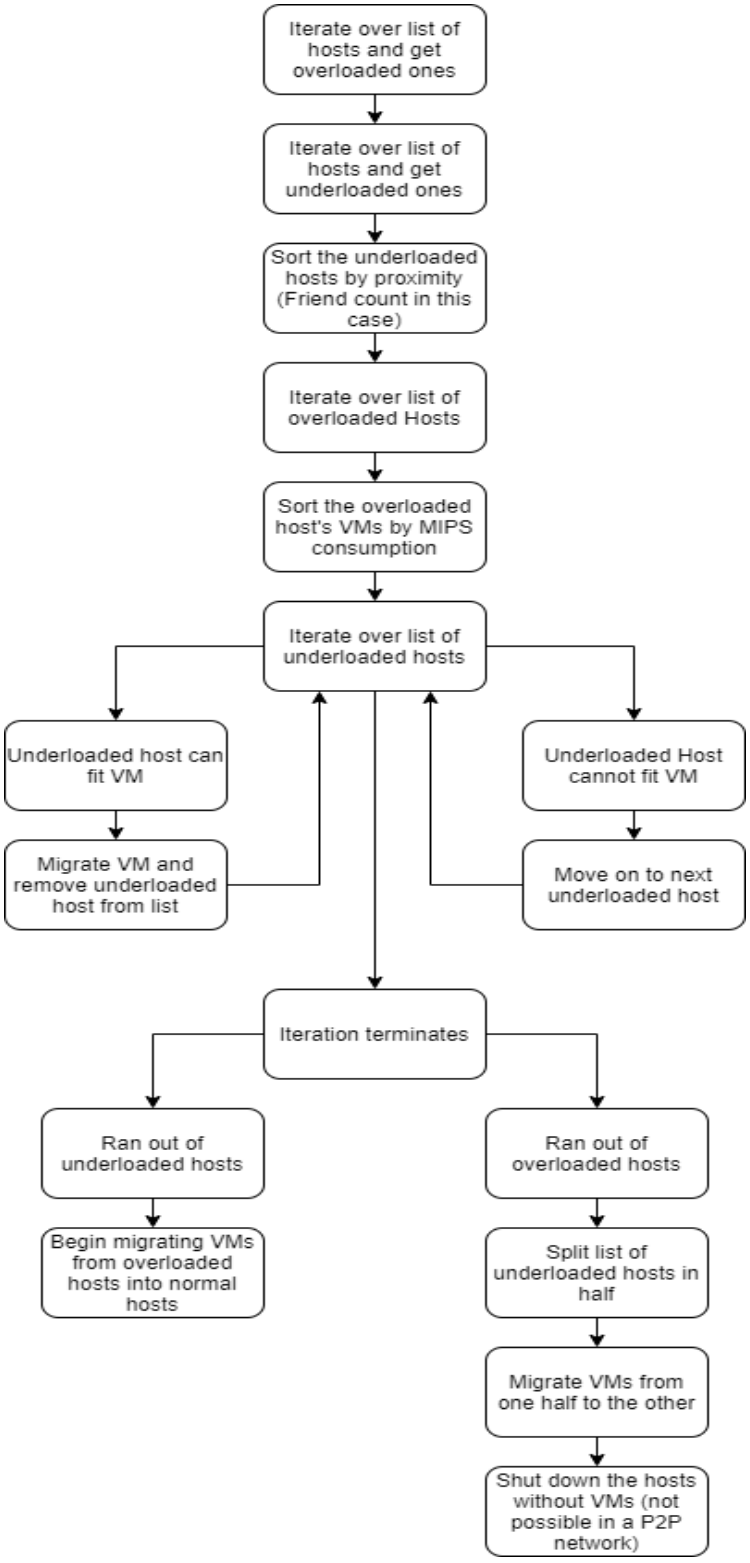

Figure 3 - EAMA Load Balancing Migration Algorithm

V. Proposed Algorithm

A new algorithm is proposed to address the issue of fairness as a primary motivation in cloud computing load balancing. Rather than treating VM migration as a necessary evil to mitigate the problems with overloaded hosts, this algorithm instead treats it as an actively helpful tool to maintain the fairness of a system and ensure a consistent user experience for those on the P2P social cloud network. This algorithm is the Social Credit algorithm, which uses a currency-like system to track the amount of effort and reward that a user is experiencing as a combination of both a host in the datacenter and the producer of a jobs to the datacenter. Instead of treating it like a marketplace however like in [6], it opts to treat it more like a trust value [16], only instead of fault rates being the key factor it uses the host's capability of NOT becoming overloaded. Social credit works in a fairly simple way, impact both the job producing end and the hosting end. On the jobs end, social credit is deducted from the user per length of the job in terms of the CPU time consumed by the job. On the hosts end, social credit is incremented per tract of CPU time used by the host. In a simulated environment, both of these values can be represented by cloudlet length.

The social credit is put to use where task migration is involved. There are two situations where a VM will be actively migrated out of a host. One is the typical case where a host is overloaded. When this occurs, the algorithm first sorts the list of VMs on the host by their social credit in descending order, and then selects the VM with the user with the lowest social credit as the unfortunate VM to be migrated out. The user then looks to the VM's job's owner's peers that are within the security level and migrates the VM over to the one with the lowest social credit.

Migrating a VM out also decrements social credit on the host that's releasing the VM and increments social credit on the host that is accepting the VM.

The other one is where a host is underloaded and also at negative social credits. In this case, the system scans the list of hosts every 10 seconds and identifies these "lazy" hosts. It will then proceed to attempt to migrate VMs with low social credit from hosts with high social credit (but are not necessarily overloaded) to each one of the lazy hosts, which will increase their social credit and decrease the social credit of the hosts which have high social credit.

The goal of social credit is not directly rewarding, but to track the amount of fairness a user is getting. A social credit score close to zero is the preferred state as it indicates high fairness.

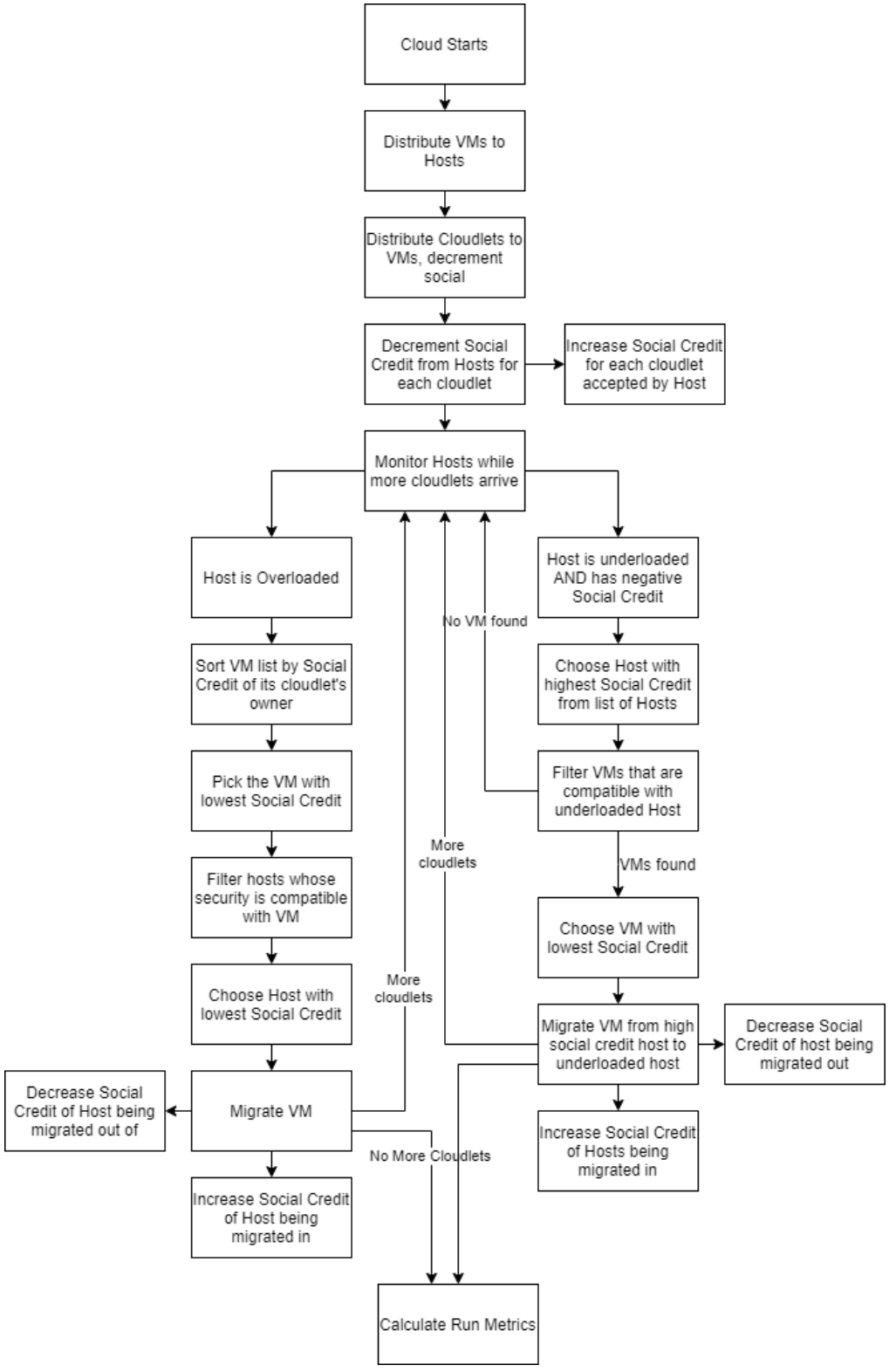

Figure 4 - Proposed Social Credit Load Balancing Algorithm

The time complexity of this algorithm is relative straight forward. The two branches of the algorithm have two complexities. For the overloaded host branch, we need to iterate over the list of hosts and detect which ones have been overloaded. Let the number of VMs in a host be $\boldsymbol{V_m}$ and the number of total hosts be $\boldsymbol{M}$. The time complexity of the first branch is $\boldsymbol{O}(\boldsymbol{V_m M} + \boldsymbol{M^2})$per each host overloaded.

For the second branch, the time complexity is $\boldsymbol{O}(\boldsymbol{V_m M} + \boldsymbol{M^2})$ every 10 seconds in the current implementation within the simulation as it does iterate over the entire host list every 10 seconds to discover underloaded hosts that also have negative social credit.

As social credit isn't directly involved in the communication between nodes, the primary method of interaction between the users/machines in this social cloud is via VM migrations. VM migrations, similar to the cellular network described in [20] [21], can be done reactively or proactively. A reactive migration occurs as a result of a host overloading, while a proactive migration occurs as a result of a low credit host underloading.

The resultant migration patterns, put into the form of a graph, can be said to be a fusion of the social network graph $\boldsymbol{G^S}(\boldsymbol{U^S}, \in^{\boldsymbol{S}})$ where $\boldsymbol{U^S}$ is the list of users and $\in^{\boldsymbol{S}}$ is the list of friend relationships between the users. And the migration "graph" that occurs whenever overloaded hosts are discovered and every 10 second window where the proactive migrations occur, as represented by $\boldsymbol{G^M}(\boldsymbol{U^M}, \in^{\boldsymbol{M}})$, where $\boldsymbol{U^M}$ is the list of Users/Hosts (the distinction isn't relevant here) that are migrating VMs in/out and $\in^{\boldsymbol{M}}$ represents the list of all possible migrations that can occur during a migration check, which is generally a spidering connection from each overloaded/high social credit host to each of the low social credit hosts.

Mathematically speaking, the social graph is an unweighted graph (as there is only one linkage option: the friend). However, given the nature of migration, the social network graph and the migration graph is not a cohesive entity like the example in [21], but rather broken up into individual slices for each VM being migrated. For a particular VM that's assigned to be migrated out, the $\boldsymbol{G^S(U^S, \in^S)}$ solely consists of the VM's owner host and the network of hosts that are reachable by a number of hops equal to the VM's current security level. While the $\boldsymbol{G^M(U^M, \in^M)}$ is the VM's current host in relation to all of the hosts which are candidates for migration. $\boldsymbol{G^S}$ is an unweighted graph, while $\boldsymbol{G^M}$ is a graph with an extremely simple layout whose edges are weighted by the difference in social credit between the source host and the various destination candidates. The union of these two graphs constitutes the resultant social migration graph for each VM, and the algorithm aims to optimize two values: Maximizing the weight between the source host and the destination host, and minimizing the social credit of the VM being migrated out. Both of these factors combine to ensure both efficient migrations for overall system performance as well as minimizing the absolute value of the social credit of all machines, which is the end goal here to maintain fairness.

Algorithm 1 – Overloaded Case

1: Iterate over List of overloaded hosts and select Host H_0:

2: Determine $\boldsymbol{G^M}(\boldsymbol{U^M}, \in^{\boldsymbol{M}})$ for H_0

3: Sort list of VMs on H_0 by their owner's social credit in ascending order

4: While H_0 is overloaded, iterate through the list of VMs and select V_S

5: Determine $\boldsymbol{G^S}(\boldsymbol{U^S}, \in^{\boldsymbol{S}})$ for V_S

6: Calculate $\boldsymbol{G^S} \cap \boldsymbol{G^M}$ with a result list of L nodes

7: Sort L by social credit of the host in ascending order

8: Pop the first host in L and migrate V_S into it

Algorithm 2 – Underloaded Migration every 10 seconds

1: Iterate over List of hosts which have negative social credit

2: Select each host that is below the underloaded threshold into list *L1*

3: Sort *L1* by the host's social credit in ascending order

4: Iterate over *L1* with iterator *H*

5: Sort list of all positive social credit hosts by their SC in descending order -> *L2*

6: Pop the first host from *L2* -> H_0

7: Sort list of VMs on H_0 by their owner's social credit in ascending order

8: Pop the first VM in this list -> V_S

9: Determine $\boldsymbol{G^S}(\boldsymbol{U^S}, \in^{\boldsymbol{S}})$ for V_S

10: **If** *H* is not within $\boldsymbol{G^S}$

11: Goto Step 8

12: **Else**

13: Migrate V_S into H_0

CloudSim Environment

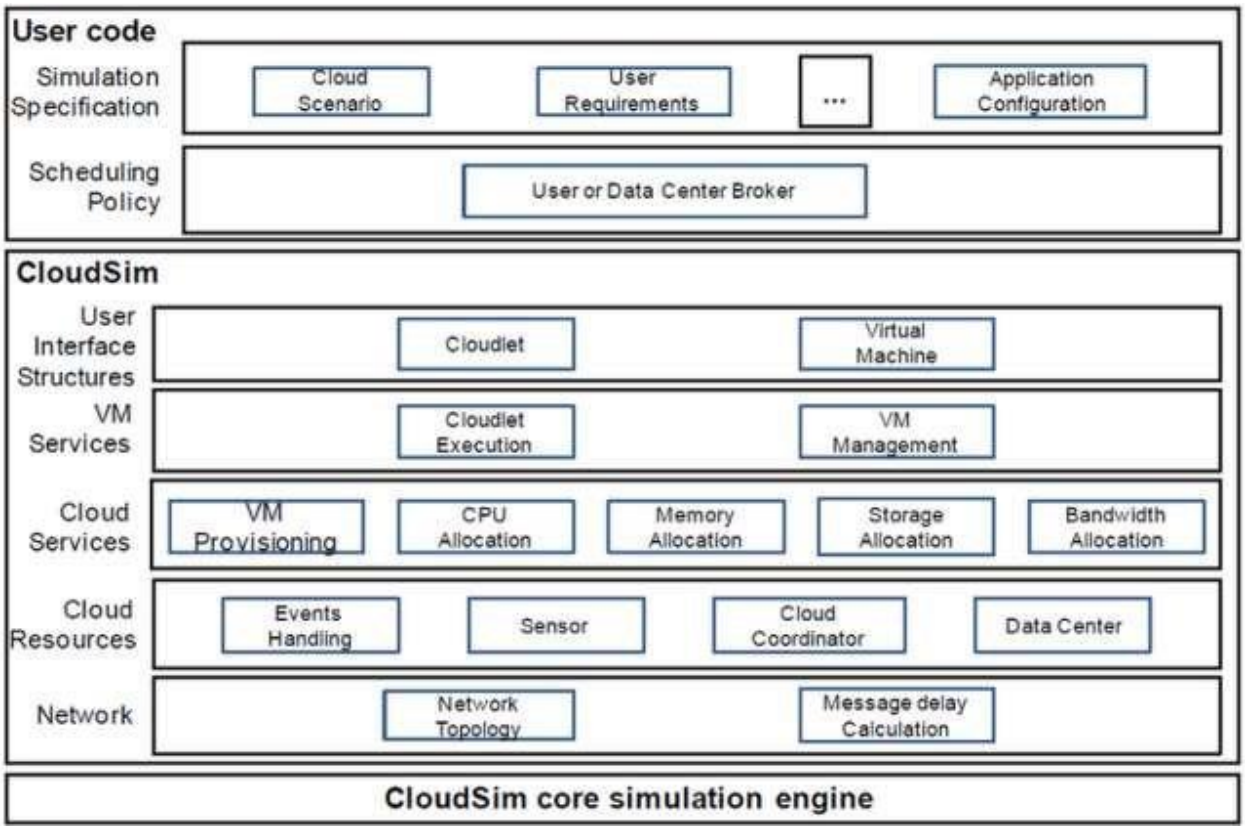

Figure 5 - Cloudsim Engine [19]

CloudSim Plus [1], maintained by Manoel Campos is chosen as the platform used to simulate the Social Credit algorithm and the algorithms used as comparison. CloudSim Plus is a Java-based simulation framework forked from the original CloudSim project with the aim of achieving cleaner code and better compatibility between the components of the simulation. In addition to that, CloudSim Plus also contains new features such as a better migration methodology and an easier power model, both of which were highly relevant in the experiment.

A CloudSim simulation [1] operates from the top-down level. Starting from the top, the main simulation engine is instantiated, which contains all of the code for running each clock tick of the

simulation. A Datacenter object is the next layer down, which represents the virtual collection of hosts and the coordinating agent that is responsible for managing them. The Datacenter object is instantiated with knowledge of the simulation and two other elements: A list of simulated Hosts, which represent the physical computational devices of the Datacenter, and a VM Allocation Policy, which is responsible for assigning VMs to hosts throughout the simulation, which includes migration. A Datacenter Broker object represents the stream of jobs and virtual machines that are assigned to the datacenter, and performs the production of VM objects and Cloudlet objects at the start of the simulation and potentially throughout. Interestingly enough, the Datacenter Broker is not coupled with the Datacenter object. The Datacenter object assigns VMs to Hosts, and Cloudlets to VMs (or they could be bound on creation), and the simulation of a living cloud computing network begins from there.

VI. Experiment Setup

I. Implementation of Users

User Count	Total Edges	Connections Per User (Average)	Hosts Per User
100	600	6 Friends	1

Table 1 - Implementation of Users

II. Implementation of Hosts

PEs	MIPS per PE	RAM	Bandwidth	Power Cost of Host at Full Load	Storage	Number
3	1000	60000 MB	32000 Mbps	140 Watts/sec	1000000 MB	100

Table 2 - Implementation of Hosts

III. Implementation of VMs

PEs	MIPS	RAM	Bandwidth	Number
1	1000	10000 MB	6400 Mbps	120

Table 3 - Implementation of VMs

IV. Implementation of Cloudlets

Length	Output/File Size	Number of Cloudlets	Security Level	Cloudlets per User
20000	300	3849	1-5	25-50

Table 4 - Implementation of Cloudlets

V. CloudSim Implementation

For the initial setup, 100 users were instantiated with an average of 6 connections per user, equivalent to six friends per user, roughly, with variance generated by a random generator seeded with "127" to produce consistent results between the three runs done with each of the algorithms. 100 hosts are then instantiated and assigned to each of the Users to represent the personal devices of the Users that composes the P2P Social cloud. 120 VMs are then instantiated and assigned to each of the hosts on a First-Fit algorithm, with their initial owning user being the Administrator who is treated as having unlimited social credit and is a direct friend of everyone. A "Cloudlet-chain" is then initialized for each user, with a total cloudlet count ranging from 25 to 50 to represent different lengths of the total amount of jobs for each User, which are then fed sequentially into the broker per user (the chains are submitted simultaneously).

To avoid touching upon CloudSim's internal workings as [1] recommends. The social cloud simulated infrastructure was constructed around the classes without replacement. A whole chain of VmSocial, HostSocial, CloudletSocial, etc. classes were constructed to support the security measures implemented by a social cloud network. The integration goes all the way up to the

three VmAllocationPolicy inherited classes that represent each of the algorithms, where the functional aspects of the security measures come into play, as every migration and allocation are filtered to restrict the list of migratable hosts to only those within the security parameters allowed by the job within the migrating VM. Alterations were mostly done in the allocation map portion of the class, but the social credit algorithm also implements an additional clock tick listener that induces a migration from a high social credit host to a low social credit host every 10 seconds within the simulation.

The values for the VMs, Hosts and Cloudlets are all based on the sample values given in the CloudSim Plus Examples [1].

VII. Result and Analysis

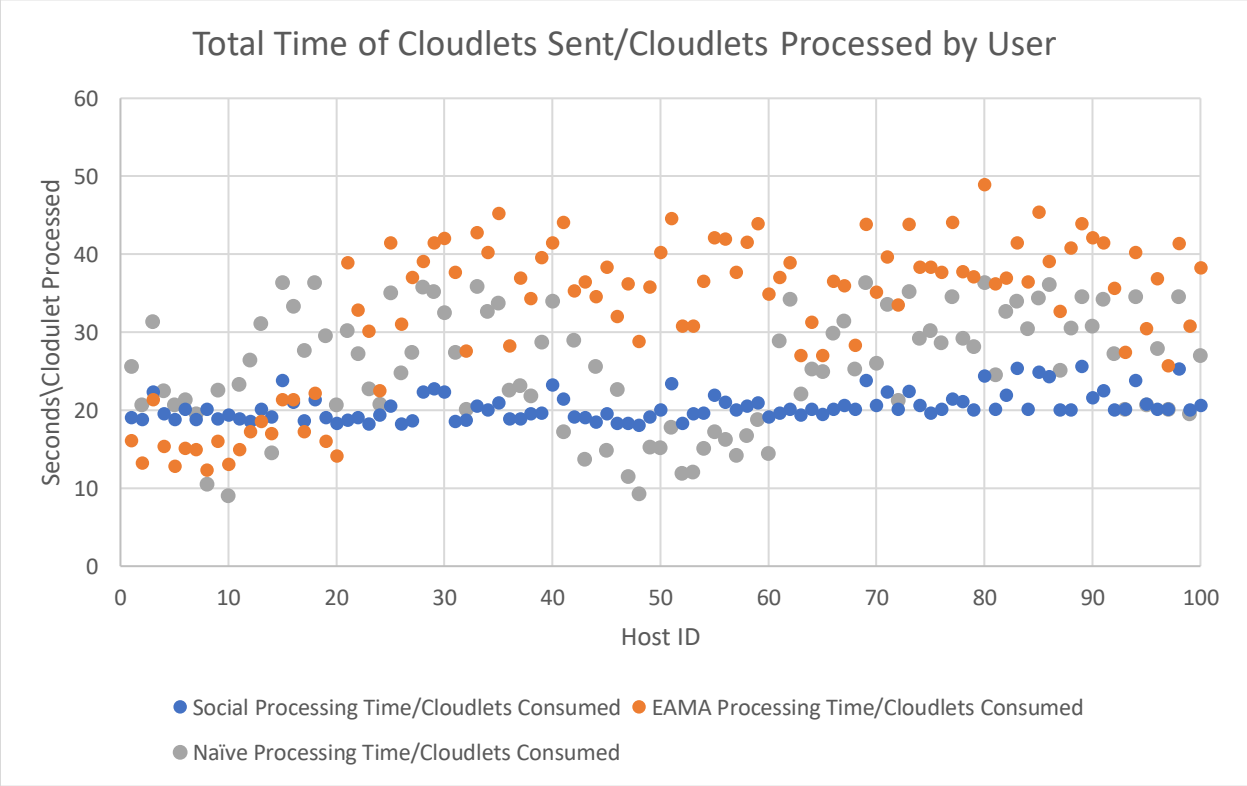

Figure 6 – Test Result in Total Cloudlet Time/Processed Count

As it is visible here, the reward a user gets, as measured by the ratio between the total amount of time the user's cloudlets take to process through the system and the total amount of other people's cloudlets processed by the user varies wildly for the Naïve and EAMA algorithms but remains very consistent for the Social Credit algorithm. This indicates that the Social Credit algorithm, which is setup with fairness in mind, accomplishes its goal. It also illustrates that on average, the Social Credit algorithm also achieves lower amounts of cloudlet time for the users, indicating overall better performance.

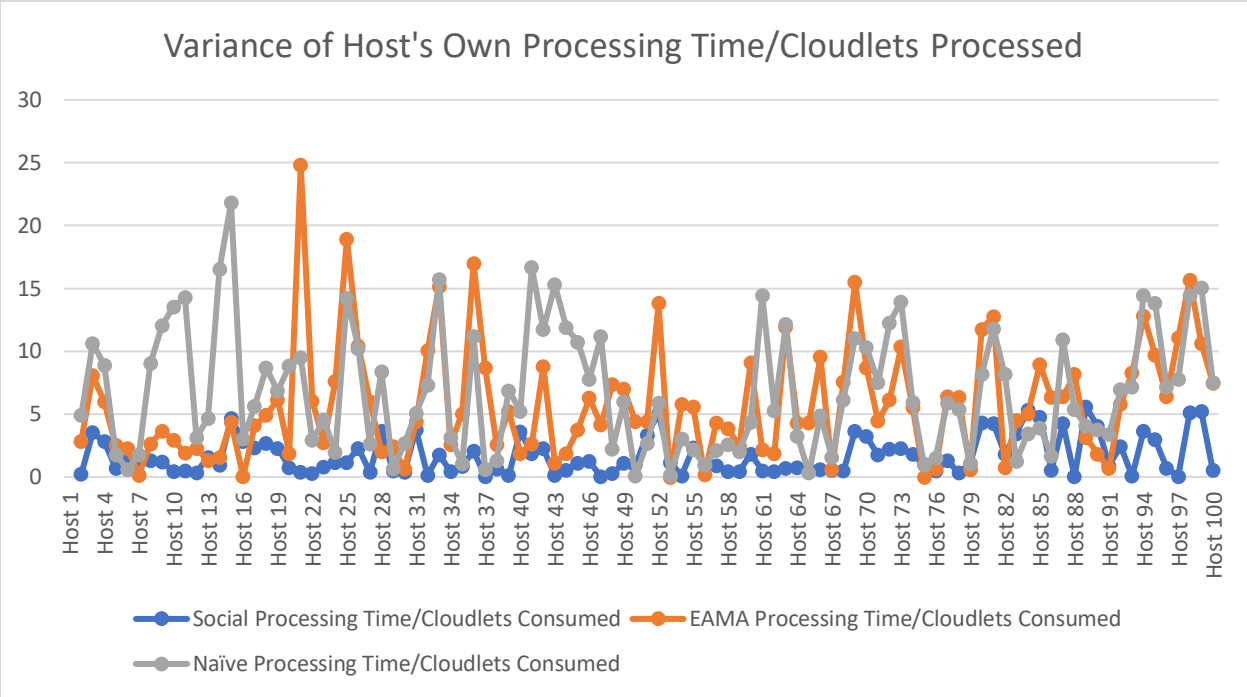

Figure 7 - Variance of Cloudlet Time/Processed Count

The variance plotted here represents the differences between each adjacent host on the dataset. Looking at this, it is clear that the variance is lower for the Social Credit algorithm compared to Naïve and EAMA. This shows that the reward for the users is more consistent for that algorithm compared to the others, and thus fairer.

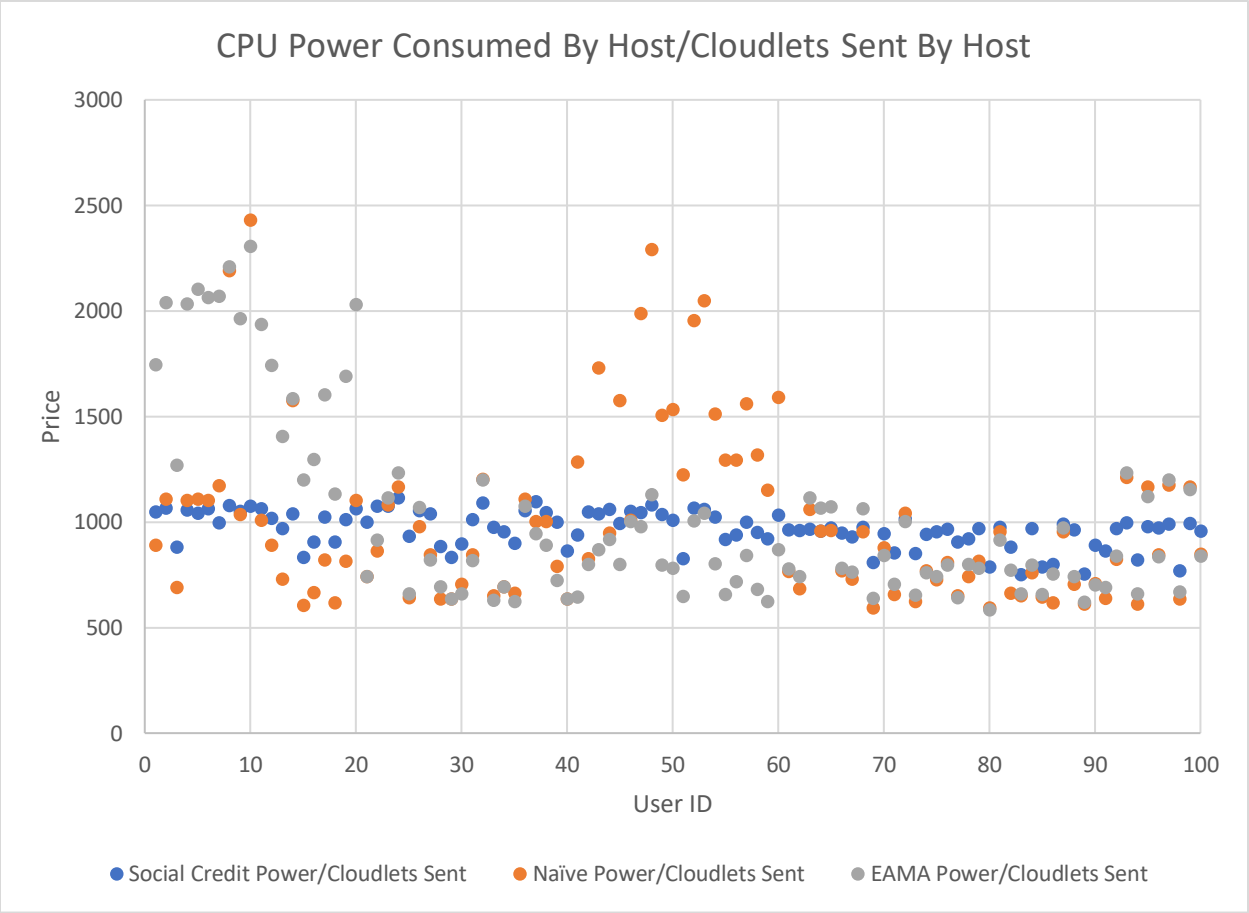

Figure 8 - Test Result Plot of CPU Power/Cloudlets Sent

Likewise for the price a user pays, as measured by the amount of CPU power the user expends towards computing for the cloud divided by the number of cloudlets the host hands over to the cloud to process. The algorithms not made with fairness in mind varies wildly and the Social Credit algorithm succeeds in maintaining a fair amount of consistency.

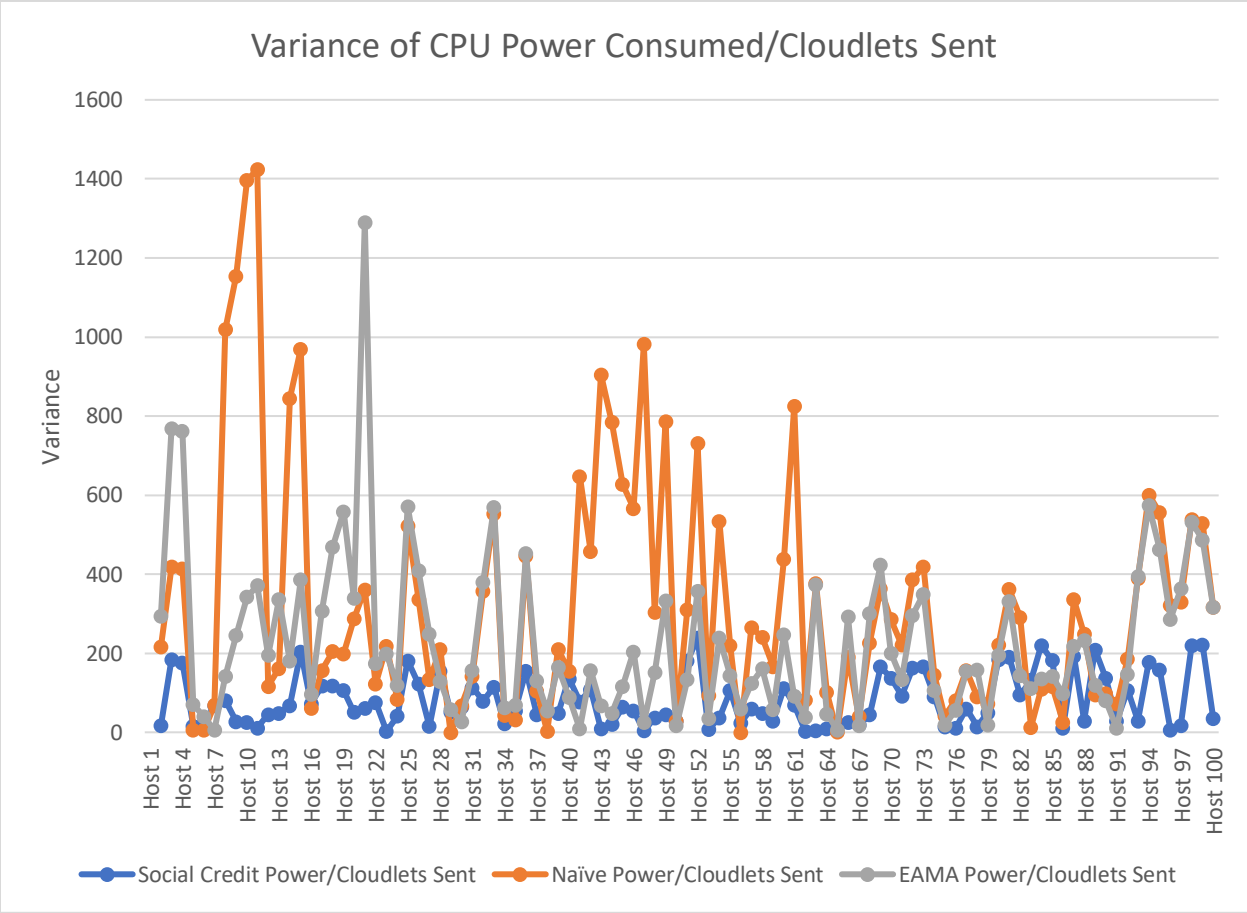

Figure 9 - Variance of CPU Power/Cloudlets Sent

The variance (the absolute difference between each neighboring data point) graph here further demonstrates a decreased amount of volatility in terms of the fairness between hosts/users.

	Social Credit	**Naïve(Best Fit)**	**EAMA**
SD of Power/Cldts Sent	**85.9758**	**402.2495**	**438.0688**
SD of Time/Cldts Processed	**1.7854**	**7.4818**	**9.6366**

Table 5 - Standard Deviation of Fairness Test Results

The standard deviation of the respective datasets from processing the 3849 cloudlets are provided above.

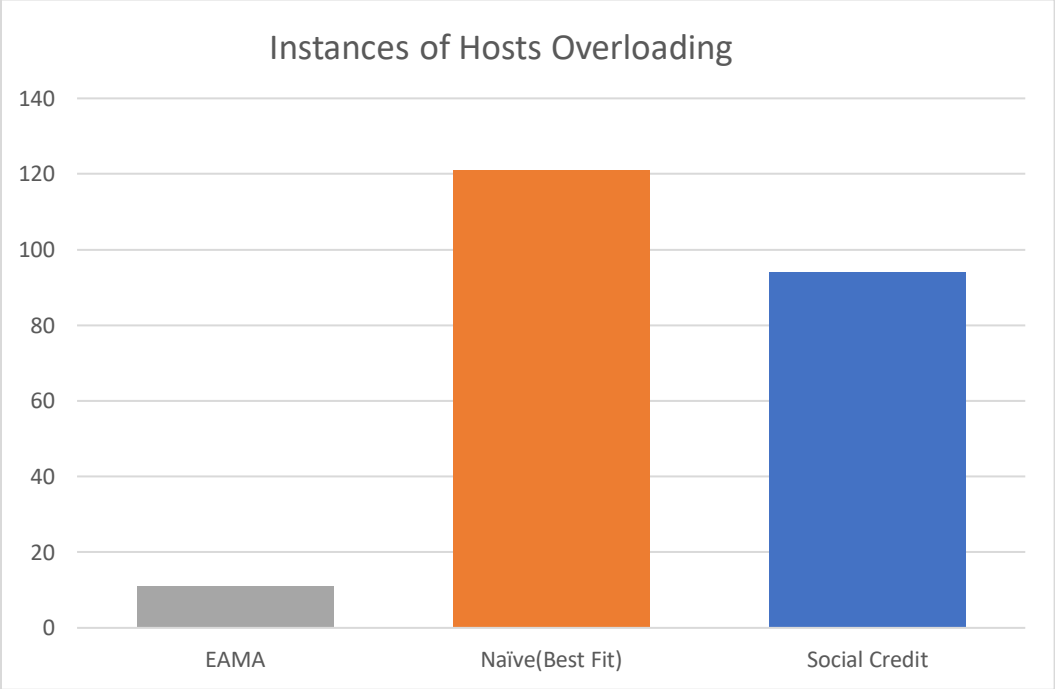

Figure 10 – Instances of Hosts Overloading during Test

While the Social Credit algorithm manages to reduce the amount of overloaded hosts compared to the naïve approach, it still cannot beat out an algorithm which is dedicated to resolving this issue. EAMA dramatically beats out the other two algorithms in reducing the amount of hosts that are overloaded through its optimized migration algorithm.

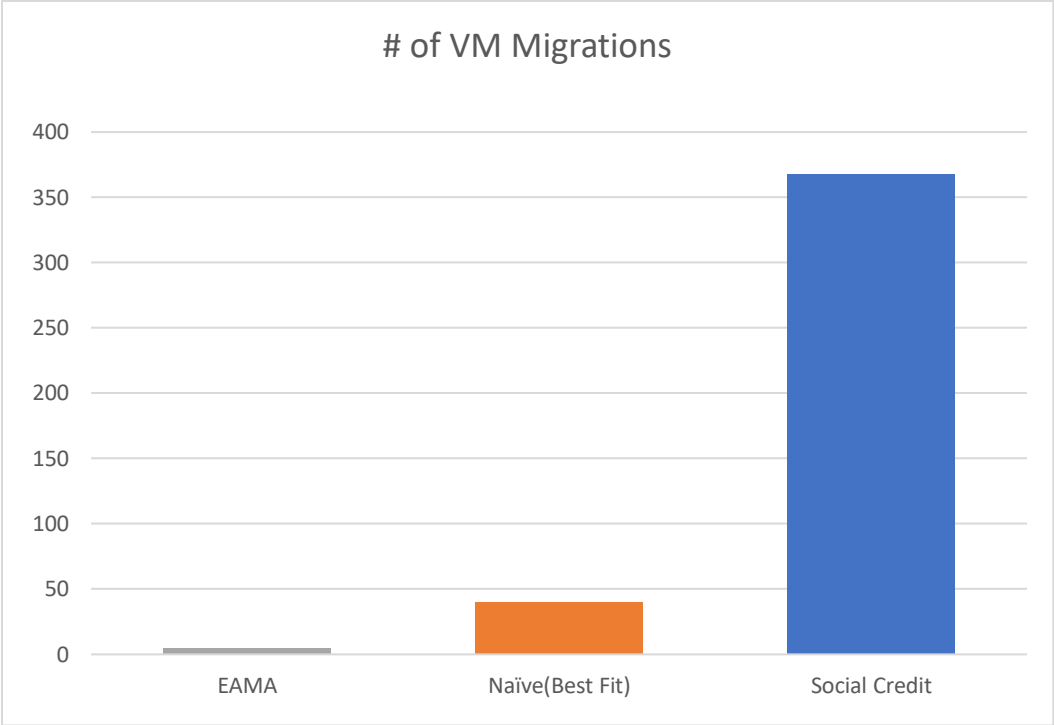

Figure 11 - Number of Migrations incurred during Test

This is an expected price however for the increased fairness and decrease in amount of overloaded hosts: The Social Credit algorithm causes a vastly increased amount of VM migrations due to the pre-emptive migrations that occur outside of hosts overloading due to fairness concerns. As the algorithm shuffles VMs to low social credit hosts every 10 seconds, it is guaranteed to incur many more migrations than the algorithms which only migrate when a host is overloaded. Also as expected, EAMA has dramatically less migrations than the other two.

Further experiments with relaxing the fairness constraint for the proactive migration in the Social Credit algorithm reveal an intuitive result: The number of migrations noticeably decreased while the standard deviation for the fairness metrics noticeably increased.

Threshold for Underloaded Migration	SD of Price	SD of Reward	Migration #
0	**85.9758**	**1.7854**	**368**
-1	**118.4236**	**2.5345**	**274**
-2	**151.9044**	**3.3353**	**198**

Table 6- Effects of Loosening Fairness Restraints

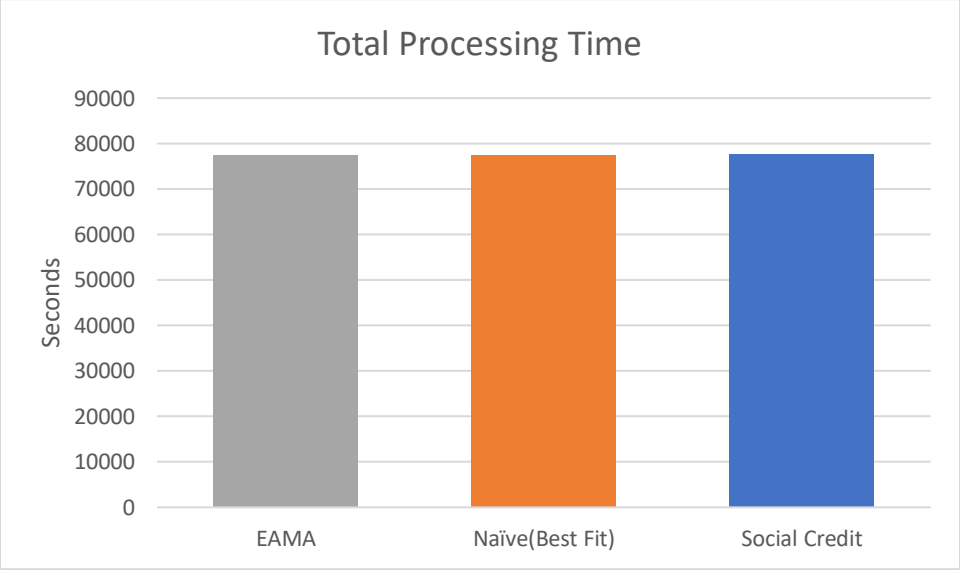

Figure 12 - Total Processing Time of all Cloudlets in Test

The total processing time, measured as the sum of every cloudlet's time from the beginning of processing to completion, between the three algorithms are largely identical. There is no overall performance tradeoff in this metric between the three of them. This is happening in spite of the greatly varied migration number between the three algorithms for two reasons. One is that the

migration overhead in CloudSim is very small, at 100 MIPS, and the migration process only takes 5 seconds to execute with the bandwidth provided, so it is using only processing capacity that is currently idle. The second one is because in CloudSim the jobs are continuing to be processed while the migration is happening, meaning that migrations don't really eat into the total amount of time the cloudlets spend being ran, and only really affects bandwidth consumption and overhead.

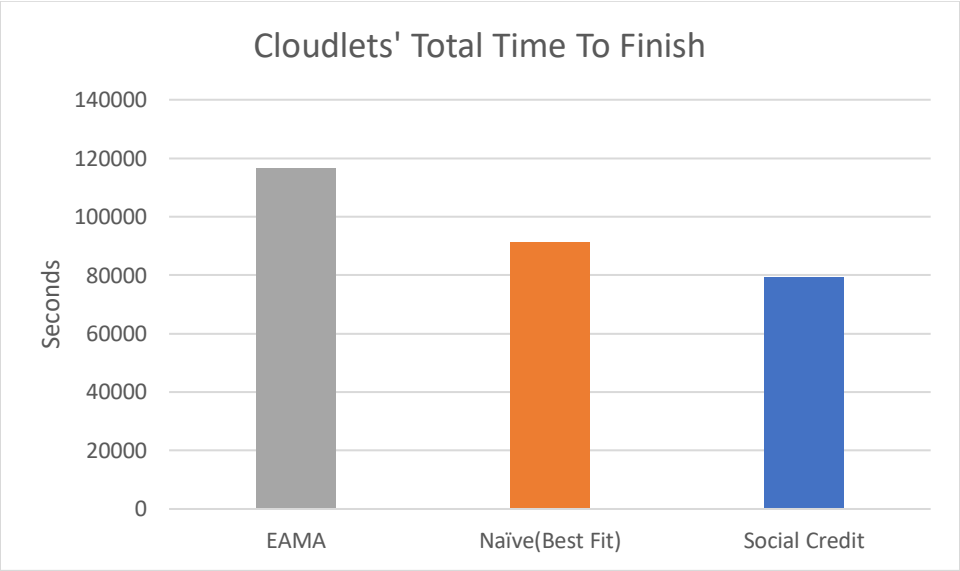

Figure 13 - Total Actual Time of Cloudlets in Test

The total amount of time cloudlets spent in the system, however, tells a much different story.

For the Social Credit algorithm, the value is close to what the processing time is, meaning that cloudlets do not spend much time in queue. The naïve approach adds roughly 15000 seconds to the total processing time, indicating that the cloudlets are spending roughly 15% of their time waiting. EAMA, however, adds a massive 40000 seconds, meaning that cloudlets are spending over 36% of their time waiting in queue, which is extremely inefficient from a time standpoint.

Overall, the results show a massive improvement in system fairness and a moderate improvement in the total amount of time spent for the cloudlets when switching from one of the conventional algorithms to the new Social Credit algorithm. There is a definitive trade-off to this, however, as the accomplishing the above objectives require active load-balancing to be done after the initial distribution of VMs to accommodate for unpredictable workloads, and such active load-balancing by necessity involves task migration. Furthermore, the restrictive nature of a social cloud and its security requirements involve necessarily suboptimal migrations as cloudlets cannot always go to the most underloaded or the least credited host in the network, meaning that proactive migrations will need more migrations to get close to the optimal configuration.

As such, the Social Credit algorithm is most appropriate for scenarios where fairness and individual host privileges are more important than migration costs such as bandwidth consumption. Setups such as a LAN-based P2P social cloud for a school or research institution where bandwidth availability is both high and low-cost can make use of it to guarantee fairness for the opt-in members that use the computational cloud. For setups across the internet, however, where distances are long and bandwidth usage incurs relatively high costs, the high amount of migrations generated by the social credit algorithm makes it untenable.

VIII. Conclusions and Future Work

For the purposes of exploring the new frontier that is P2P cloud computing, this paper focuses on improving the metrics of a part of cloud computing that is non-existant without P2P: The fairness for a user when they take on both the role of consumer and producer. The goal is to sought out an improvement in fairness in terms of CPU and processing time, and to that end the paper has reached a conclusion on a viable solution for that in the Social Credit algorithm. Further improvements on this research are planned. For starters, the costs upon the host's RAM and bandwidth should be added in, as well as a more restrictive and random set of parameters for them to better simulate the realities of a campus P2P network where hosts are personal computers with no pattern to their hardware specifications, or campus equipment that are likely not uniform in composition. Costs upon RAM, bandwidth and potentially storage make up a significant portion of datacenter expenses, and its associated costs should naturally also extend to P2P networks. There is also a potential in further expanding the exploration of fairness in a P2P cloud system. As the concept of resource sharing in a peer-to-peer environment with relative strangers is comparatively untested, there would be more and different metrics to measure fairness in a cloud system compared to the currently simple definitions of price and reward. The Social Credit algorithm in particular could use more testing against different load balancing algorithms and potentially algorithms that also take fairness into account to properly compare its performance. Machine-learning based predictive algorithms as a standalone comparison or as an enhancement to Social Credit, as it can facilitate fairness increases during the assignment phase instead of relying on migrations to increase fairness.

References

[1] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire. CloudSim Plus: a Cloud Computing Simulation Framework Pursuing Software Engineering Principles for Improved Modularity, Extensibility and Correctness, in IFIP/IEEE International Symposium on Integrated Network Management, 2017, p. 7.

[2] Chard, K., Caton, S., Rana, O., & Bubendorfer, K. (2010). Social Cloud: Cloud Computing in Social Networks. *2010 IEEE 3rd International Conference on Cloud Computing*, 99–106. https://doi.org/10.1109/CLOUD.2010.28

[3] Chard, K., Bubendorfer, K., Caton, S., & Rana, O. F. (2012). Social Cloud Computing: A Vision for Socially Motivated Resource Sharing. *IEEE Transactions on Services Computing*, *5*(4), 551–563. https://doi.org/10.1109/TSC.2011.39

[4] Chang, C., Srirama, S., & Ling, S. (2014). SPiCa: a social private cloud computing application framework. Proceedings of the 13th International Conference on Mobile and Ubiquitous Multimedia, 2014-, 30–39. https://doi.org/10.1145/2677972.2677979

[5] A. Mohaisen, H. Tran, A. Chandra and Y. Kim, "Trustworthy Distributed Computing on Social Networks," in IEEE Transactions on Services Computing, vol. 7, no. 3, pp. 333-345, July-Sept. 2014, doi: 10.1109/TSC.2013.56.

[6] Babaoglu, Ozalp & Marzolla, Moreno & Tamburini, Michele. (2012). Design and Implementation of a P2P Cloud System. Proceedings of the ACM Symposium on Applied Computing. 10.1145/2245276.2245357.

[7] Falcão, E. de L., Brasileiro, F., Brito, A., & Vivas, J. L. (2016). Enhancing fairness in P2P cloud federations. Computers & Electrical Engineering, 56, 884–897. https://doi.org/10.1016/j.compeleceng.2016.05.007

[8] Joe-Wong, C., & Sen, S. (2018). Harnessing the Power of the Cloud: Revenue, Fairness, and Cloud Neutrality. Journal of Management Information Systems, 35(3), 813–836. https://doi.org/10.1080/07421222.2018.1481639

[9] Tang, S., Yu, C., & Li, Y. (2020). Fairness-Efficiency Scheduling for Cloud Computing with Soft Fairness Guarantees. IEEE Transactions on Cloud Computing, 1–1. https://doi.org/10.1109/TCC.2020.3021084

[10] Tajamolian, M., Ghasemzadeh, M. (2019). Analytical evaluation of an innovative decision-making algorithm for VM live migration. *Journal of AI and Data Mining*, 7(4), 589-596. doi: 10.22044/jadm.2018.7178.1847

[11] Ibrahim, M., Imran, M., Jamil, F., Lee, Y. J., & Kim, D.-H. (2021). Eama: Efficient adaptive migration algorithm for cloud data centers (cdcs). Symmetry (Basel), 13(4), 690–. https://doi.org/10.3390/sym13040690

[12] Li, B., Liang, S., Tian, L., Chen, D., & Zhang, M. (2020). An Adaptive Task Scheduling Method for Networked UAV Combat Cloud System Based on Virtual Machine and Task Migration. Mathematical Problems in Engineering, 2020, 1–12. https://doi.org/10.1155/2020/5391479

[13] Alam, M., Haidri, R. A., & Shahid, M. (2020). Resource-aware load balancing model for batch of tasks (BoT) with best fit migration policy on heterogeneous distributed computing systems. International Journal of Pervasive Computing and Communications, 16(2), 113–141. https://doi.org/10.1108/IJPCC-10-2019-0081

[14] Nieuwenhuis, Ehrenhard, M. L., & Prause, L. (2018). The shift to Cloud Computing: The impact of disruptive technology on the enterprise software business ecosystem. Technological Forecasting & Social Change, 129, 308–313. https://doi.org/10.1016/j.techfore.2017.09.037

[15] Ye, & Pan, J. (2011). A new P2P Campus Cloud framework. 2011 IEEE International Conference on Anti-Counterfeiting, Security and Identification, 1–4. https://doi.org/10.1109/ASID.2011.5967402

[16] Gupta, Goyal, M. K., & Kumar, P. (2013). Trust and reliability based load balancing algorithm for cloud IaaS. 2013 3rd IEEE International Advance Computing Conference (IACC), 65–69. https://doi.org/10.1109/IAdCC.2013.6514196

[17] Aslam, & Shah, M. A. (2016). Load balancing algorithms in cloud computing: A survey of modern techniques. 2015 National Software Engineering Conference, NSEC 2015, 30–35. https://doi.org/10.1109/NSEC.2015.7396341

[18] Wu, & Wu, G. (2009). Reputation Mechanism in Peer-to-Peer Network. 2009 First International Conference on Information Science and Engineering, 1793–1796. https://doi.org/10.1109/ICISE.2009.882

[19] Singh, A. (2021, January 21). *CloudSim Simulation Toolkit: An introduction*. Cloudsim Tutorials. Retrieved November 13, 2021, from https://www.cloudsimtutorials.online/cloudsim-simulation-toolkit-an-introduction/.

[20] N Saxena, FH Kumbhar, A Roy, "Exploiting social relationships for trustworthy D2D relay in 5G cellular networks", *IEEE Communications Magazine*, 58 (2), pp. 48-53.

[21] FH Kumbhar, N Saxena, A Roy, "Social Reliable D2D Relay for Trustworthy Paradigm in 5G Wireless Networks", *Peer-to-Peer Networking and Applications*, pp. 1-13