Journal of Theoretical and Applied Information Technology
30th November 2020. Vol.98. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OPTIFOG: OPTIMIZATION OF HETEROGENEOUS FOG COMPUTING FOR QOS IN HEALTH CARE

[1]PRATIK KANANI, [2]DR. MAMTA PADOLE

[1] Assistant Professor, Dwarkadas J. Sanghvi College of Engineering, University of Mumbai, India

[2] Associate Professor, CSE Dept, The Maharaja Sayajirao University of Baroda, Gujarat, India

E-mail: [1]pratikkanani123@gmail.com

ABSTRACT

A patient's life can be saved if it is possible to make quicker decisions based on faster processing of real-time health care data, such as ECG processing. To achieve faster decision making, contemporary health care applications use cloud computing for such data. When cloud computing is used, data transmission deferrals may cause delays in the decision-making process. To overcome this, Fog Computing is used. Fog Computing saves energy, bandwidth and prevents transmission latencies but, lacks in computing power as compared to Cloud Computing. To enhance the computing power of the Fog node, a Cluster of Raspberry Pi having heterogeneous configurations can be used. In Health Care applications the Fog Computing performance can be assessed by measuring the time elapsed between the generation of the health care data and decision-making. In this paper, ECG signal analysis is taken as a processing job in Fog Computing. Dispy is used to facilitate the scalability and parallel data processing on a Cluster of Raspberry Pi used for Fog Computing, to enable faster decision making. Further, the performance of the Raspberry Pi cluster-systems using dispy are analyzed and optimized step by step based on different parameters. The first parameter is data transmission time which is improvised by minimizing network overheads. Other optimization parameters like CPU usage, number of cores, response time and available memory space, these parameters are considered and varied, to assess the performance of Heterogeneous Raspberry Pi cluster. Based on the results obtained, a novel optimization approach "OptiFog" is proposed to achieve faster computation in worst-case scenarios by varying and assigning jobs to the nodes to measure performance parameters in Distributed Fog Computing. Based on the obtained results "OptiFog" assures best possible improvement in the performance of the Distributed Fog Computing environment.

Keywords: *Dispy, Distributed Fog Computing, ECG Analytics, Heterogeneous Fog Computing, Optifog, Optimization In Fog Computing, Raspberry Pi Cluster*

1. INTRODUCTION

IoT sensors help in collecting patient's real-time data which is time-sensitive with regard to decision making. The volume of this data depends on the sampling frequency of data generated and the density of sensors spread over the region. This data is time-sensitive data i.e., a timely decision should be made based on it; else, the data will have no value. If this data is related to health care data then, it is more time-sensitive in terms of decision making e.g. real-time ECG monitoring data. In traditional IoT architecture, this data is sent over the cloud for processing and for storage for future reference, but the transmission delays are very high and as a result, redundant data keeps on streaming over the cloud on which real-time decision making cannot be done. An alternative to this is the Fog Computing [1] approach where data is processed in the LAN and only the significant and future referral data will be sent over the cloud. The remaining data is managed at the fog layer itself [2]. Fog Computing saves network bandwidth, redundant data transmission, and energy thus, reducing latency. So, if we give time-sensitive health care data to the fog then, it can process the data faster, as a result, it helps to avail medical facilities faster and hence preventing patient mortality. But fog lags in terms of computing power. Fog layer cannot handle many data streams due to its computational limitations. To overcome these limitations distributed computing can be used in the Fog Computing paradigm.

Raspberry Pi (R-Pi) can be used as a Fog Computing node [3]. To improve the overall computation capabilities of the Fog Computing layer, the Raspberry Pi cluster can be used. The

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Raspberry Pi cluster is a device made up of multiple Raspberry Pi, having different configurations. In this paper, the Raspberry Pi cluster is acting as a Fog Computing Node. Raspberry Pi cluster forms a distributed system on which it does processing in parallel for faster computation. In subsection of section one, Raspberry Pi and its cluster are discussed in detail.

To implement distributed computing on Raspberry Pi cluster, "dispy" is used. Dispy is responsible to make parallel processing possible across all nodes in the Pi cluster. It creates n channels for n slaves and keeps on assigning the task to each slave through these channels and waits for their reply. After getting a reply on the channel, it allocates a new task to the node and so on. Distributed computing and dispy are discussed in detail in section three.

Every algorithm gives the best possible results if it is working in the best-case scenario (Big Omega (Ω) case). But if any approach can generate the best results in the worst-case (Big Oh (O)) scenarios then it will definitely work well on average (Big Theta (Θ)) and best-case scenarios. In this paper, a comparative study/implementation of existing techniques is done and proposes an approach that works well even when all nodes of the cluster are preloaded with other computation tasks. It is being done to make sure that the proposed "OptiFog" algorithm would perform best in worst-case scenarios too. In section four, the working environment is explained in terms of its heterogeneity, hardware and software configurations. Section four also discusses an important aspect of analyzing ECG signal and their techniques in detail. This paper considers the computation task of ECG signal analysis and this analysis is done by ECG signals of varying lengths.

In section five, the dispy setup is installed on all the nodes of the cluster and ECG signal analysis is performed by master-slave architecture. In the initial run, the sub job is considering a set of two ECG waves and computation readings are recorded. After analyzing these readings, the sub job task is modified in terms of the number of ECG waves to reduce the overheads. Performance parameters for each of these cases are explained. Various logical terms, technical terms, symbols and related functions are presented and explained in section six.

In any computing system, there are many parameters on which the overall computation is relying on. In this paper, few of these parameters like CPU usage, number of cores, memory and response time are identified. The planned heterogeneous set up is run against each of these parameters and readings are recorded to understand the impact of these parameters in the given computation. After the experiment is performed, each graph representing every parameter is discussed in detail. Section seven is all about these parameters and understanding their impact on the computational task.

In Section eight, the proposed "OptiFog" algorithm is explained in detail with its design and working. The parameters used in the algorithm are explained, the expected performance outcome and performance improvement is discussed at length. In the last section, the proposed algorithm is tested on different data sets to prove its validity. The Speedup factor and the percentage of improvement is presented at the end.

1.1 Aim and Objectives

- To extend the distributed computing in the Fog Computing to support latency-sensitive IoT based health care applications
- To design the smart Fog Computing cluster with smart job allocation to satisfy the SLAs in terms of ensuring the optimal response time and resource utilization
- To develop an algorithm, to dynamically decide node capability, that enables to identify the node that can be assigned the appropriate task for health care applications in Fog Computing
- To develop an optimized Fog Computing based performance model in the health care domain to serve the community better

1.2 Focus of the paper

- Considering a Low latency health care application for delay-sensitive real-time applications
- Analyzing dispy performance in fog-distributed computing
- Analyzing and understanding other processing factors and techniques to get the best out of it
- Explaining different performing parameters and their impact on the computational task
- Developing the algorithm best suited for health care applications and Raspberry pi cluster in Fog Computing

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

1.3 Raspberry Pi and R-Pi cluster

Raspberry Pi was introduced by the Raspberry Pi Foundation and is a series of small single-board computers [4]. Although they were intended to be used for teaching the basics of computer science, now they have surpassed it. Today it can be used for everything from Home Entertainment System to Cryptocurrency Mining [5-6].

A Raspberry Pi cluster is a collection of Raspberry Pi, all running as a single computer using parallel computing [7]. Raspberry Pi Clusters use an architecture involving a Master node and Slave node, the master node is responsible for giving instructions for the tasks that need to be performed [8-9]. Experiment and research conducted in the field of cloud computing have shown that it is possible to run edge computing on these Raspberry Pi clusters [8, 10]. These clusters can have a variety of different applications from running as MQTT Broker [9], Fog Computing framework for Wearable IoT Devices [11] to teaching distributed computing [12]. The presence of multiple Raspberry Pi nodes in the system improves the reliability of the system.

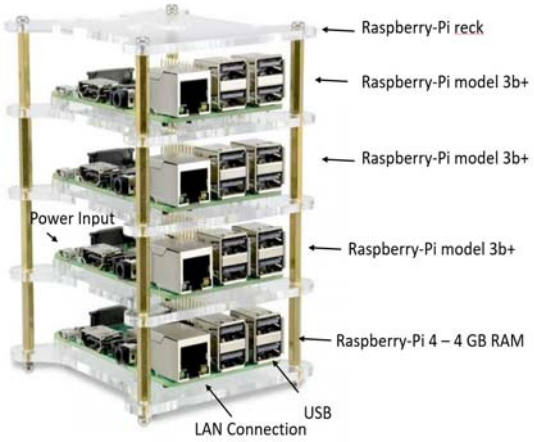

Figure 1: Raspberry Pi Cluster used in the system

2. LITERATURE REVIEW

The literature review studied is targeting the systems which have used or optimized system related to Distributed computing in terms of performance. It also focuses on the usage and improvements in the system due to memory, response time, CPU usage and the number of cores. Later it is aiming at the work related to Fog Computing in context to performance improvement.

Helen Karatza et al. [13] have addressed the issues faced while scheduling parallel jobs on a cluster of distributed processors. Two types of routing schemes are considered. Also, 3 types of scheduling techniques are considered. The objective of this paper is to analyze the performance of these task scheduling algorithms in each case of routing. Various other system parameters that need to be considered for job submission and task scheduling are also studied and tabulated in this paper. This paper analyses the feasibility of each scheduling algorithm in each routing scenario and the impact of the system parameters in task scheduling. Simulation is used to analyze the performance of the algorithms in different system load conditions. By analyzing the simulation we can understand the impact of the scheduling policies on the system performance.

Abdou Guermouche et al. [14] have proposed a system that aims at improving the working of a parallel multifrontal solver, MUMPS. This scheduling approach is memory-based. Memory constraints are used to choose a processor's slaves and/or associates. Slaves are chosen according to their memory availability. It aims at reducing the used stack size at run time. Li Xiao et al [15] have proposed a paper that tries to enhance the effective usage of global memory. Job distribution strategies are also built accordingly. When a node has insufficient memory to accept jobs, the extra load is then migrated to other associates with sufficient memory availability. Unbalanced memory allocations for jobs cause page faults, so the motive is to minimize the same thereby improving efficiency. The load sharing policy proposed improves the performance of memory-bound jobs. Yuyan Sun et al. [16] have proposed a paper in which they describe distributed systems. In distributed systems, it is essential to have a fair load-sharing policy to ensure that the computational capacity is completely utilized. The load sharing policy has a major impact on performance. The system memory has a major role in system performance. Therefore available memory becomes the base of the load sharing systems. In memory-bound jobs, memory-based load sharing system has higher performance. The developed algorithm shows better performance than FCFS and Round Robin algorithm in load sharing systems. The memory-based load sharing systems are more adaptive in terms of performance and they are sensitive towards the memory variance. Kizhakkethil et al. [17] present a memory-based hybrid Dragonfly algorithm for optimization.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Nawwaf Kharma et al. [18] have proposed H2GS which is a two-phase scheduling algorithm. It works on distributed systems, while the main focus is on heterogeneous systems. A highly efficient schedule is generated using a heuristic list-based algorithm that makes up phase one. In phase two shorter schedules are evolved. Tasks to be scheduled are given priorities. The ready task with the highest priority is selected for scheduling. The next phase is where the processor is selected. Here a task is selected and submitted to the processor to minimize the time of execution. In the paper presented by Haluk Topcuoglu et al. [19], an algorithm for scheduling is implemented on processors whose numbers are predefined. The motive is to meet efficient scheduling and enhanced performance simultaneously. The name of the algorithm is Heterogeneous Earliest Finish Time (HEFT). At each step, the maximum upward rank value is chosen and assigned to a processor. Based on an insertion-based approach, the earliest completion time is minimized. HEFT algorithm is robust and performs well over a wide range of graph structures.

Bao Liu et al. [20] have presented several scheduling/co-scheduling techniques employed in distributed systems. Predictive scheduling and Proportional-sharing scheduling are the types of local scheduling introduced here. Predictive scheduling provides adaptivity, intelligence, and proactivity to adopt new architectures and changes in the environment automatically. It learns new architectures, algorithms, and methods that are embedded in the system. The allocator aggregates previous inputs, in the form of a vector of performance information (CPU usage), into sets. Each set corresponds to a scheduling decision. Sets are split or merged, to keep a limited memory demand, by the allocator. Marjan Khosravi Talebi et al. [21] have presented an algorithm that is used for scheduling in the cloud computing environment. In this algorithm, parameters like processor status are used to obtain the node on which the job should be scheduled. The goal is to obtain an efficient scheduling method that minimizes the overall processing time of all the loads by distributing the loads amongst all the available processors. The processor to which the current job has to be assigned is decided by a formula that takes into account the history of scheduling along with the processing power and link time.

Alfredo Goldman et al. [22] have presented an article about the different scheduling algorithms used in distributed computing in cloud computing. The prominent difference between distributed computing and cloud-based computing is the incorporation of virtualized systems. Virtual Machine (VM) allocation is performed to strengthen the server. Here, the scheduler pool is denominated as the software being considered for scheduling. Hardware requirements (number of cores in the system and their usage statistics, etc.) are used by the scheduler for scheduling. Zafeirios Papazachos et al. [23] have studied various gang scheduling algorithms and analyzed their efficiency for clusters consisting of multi-core systems. Gangs are scheduled in multi-core cluster systems using the suggested migration structure. An evaluation model provides results on the performance of the system. In gang scheduling, fragmentation is caused when the size of the gangs prevents them from fitting in idle cores. Flexible and adjustable schedules are made by dynamically migrating parallel jobs. Migrations are given importance as it satisfies the requirement of load balancing.

Salim Bitam et al. [24] have focused to develop a job scheduling task for mobile users in Fog Computing. They have used the Bees Swarm algorithm with CPU execution time and the total amount of memory. One of the important aspects is to save the network bandwidth in communication, Frank et al. [25] have suggested to compress the raw data and resend it, and decompress it for processing on the receiver side. But, this compression and decompression save the network bandwidth but increases response time in health care which is very critical.

3. DISTRIBUTED COMPUTING AND DISPY

Distributed Systems are a group of independent computers that connect to offer various services. They share and store data without physically sharing memory or processor with other computers. These machines have a shared state, operate concurrently and can fail independently without affecting the whole system's uptime. A distributed computer system comprises multiple software components that run on multiple computers but act as a single machine. A distributed system allows reduced computational time over massive datasets [26]. Distributed systems are classified into two major categories: centralized (client-server) and decentralized (peer to peer) system. To facilitate distributed computing, dispy is a very good tool. The dispy tool is developed in python and it is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

available for different needs and greed. The dispy tool [27], is available for different distributed computing tasks.

3.1 Why dispy?

To implement the distributed computing for Raspberry Pi cluster in Fog Computing "dispy" is selected. Dispy is developed in python and python works very well with Raspbian operating systems. The main problem faced in deployments of distributed computing is that each slave node has to be configured for a particular application context. If more slaves add-up then it needs more configuration. This makes scalability in distributed computing a bit difficult, but in case of dispy only master node has to be configured and on all slave nodes, only dispy should be installed. No need to configure every slave. This makes a distributed system more scalable if we use dispy.

4. UNDERSTANDING THE WORKING ENVIRONMENT

To perform the experimentation different hardware and software configurations are chosen. Here, we will understand the need for different configurations with their specifications. Each node in the system is preloaded by some computation and the processing job with its details are discussed here.

4.1 The need for Heterogeneous Configuration

In this system, two setups are used. In both the setups we are varying the number of nodes in the cluster from 1 to 4. This is to understand the effect of computation-distribution by techniques like memory, response time, CPU usage and the number of cores. And we have taken two types of nodes in the system and maximum nodes are four in the cluster. In one setup all 4 nodes are homogeneous in terms of its hardware. But, in another setup three homogeneous and one heterogeneous node is taken. This is to see the effect of processing, load distribution, overall performance effect and adaptiveness of the algorithm in a cluster, in different configurations and parameters.

4.2 Hardware Configurations of the cluster nodes

For the projected system, Raspberry Pi 3 model b+ [28] and Raspberry Pi 4 [29] models are used. The hardware configurations are as follows.

Table 1: Hardware configurations of cluster nodes

Hardware Module	Raspberry Pi 3 Model b+	Raspberry Pi 4
Processor	Broadcom BCM2837B0, Quad-core Cortex-A53 (ARMv8) 64-bit SoC	Broadcom 2711, Quad-core Cortex-A72 64-bit SoC
Operating Frequency	1.4 GHz	1.5 GHz
Bluetooth	4.2	5.0
Wi-Fi	2.4 GHZ / 5.0 GHZ IEEE 802.11.b/g/n/ac/wireless LAN	2.4 GHZ / 5.0 GHZ IEEE 802.11.b/g/n/ac/ wireless LAN
Memory	1GB LPDDR2 SDRAM	4GB LPDDR4 SDRAM
SD Card Support	Micro SD card	Micro SD card
Operating voltage & current	DC 5V/2.5A DC	DC 5V/3A

The Raspberry Pi 4 node is having a higher hardware configuration than the Raspberry Pi 3 b+ model. The Pi 4 node is higher in terms of processing, communication, and memory. Purposely the higher node is introduced in the system so that the system behavior can be studied over the other parameters and a good algorithm can be designed accordingly. And in all varying nodes, the dispy server node is always Raspberry Pi 3 b+. Other computing parameters on which the system performance relies are bus speed, internal clock rate, instruction set architecture, cache memory and processor. The details are given below in table 2.

Table 2: Number of nodes and nodes selection

Experimental Configuration	Number of R-Pi 3	Number of R-Pi 4
1 node	1	-
2 nodes	2	-
3 nodes	3	-
4 nodes	4	-
4* nodes	3	1

Thus, in our setup, cluster 1 comprises of all 4 nodes which are Raspberry Pi 3 b+ models and cluster 2 comprises of 3 nodes which are Raspberry Pi 3 b+ model and 1 node which is Raspberry Pi 4 model.

Journal of Theoretical and Applied Information Technology
30th November 2020. Vol.98. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.3 Continuous Load on cluster nodes

To design the best suitable algorithm to process health care data efficiently in the Fog clustering environment. All cluster nodes are kept busy in some or the other computational work apart from health care data processing. This computational task keeps cluster node occupied till a certain level of CPU usage and with that, the node is allowed to perform the health care data processing task. To create the system load, the "stress" tool is used in the Raspbian operating system environment. The "Stress" is a tool [30] used to test system performances when they are loaded. System admin uses this tool to see the performance of I/O syncs, VM status, cache thrashing, CPU usage, driver performance, and process creation and termination. This tool can generate different sorts of load on the system as specified in the option field. And it can continue generating that much stress on the system till the said time ends. Here, the following command is used to generate the CPU load for the needed time.

The "**sudo stress –cpu 1 –timeout 20000**" command which keeps CPU busy for nearly 25% for 20000 seconds, till we run and test all algorithms in the system.

4.4 Processing Job Description and logic

ECG is a periodic wave [31], it repeats its cycles after a certain interval of time. It has P-Q-R-S-T-U points as their reference points representing respective peaks. Now the important part is to get PR, QRS and QT intervals out of these waves for each and every wave.

Table 3: Standard ECG Intervals for a healthy adult with standard bpm

Intervals	Normal Value	Normal Variation
QT Intervals	400 ms	±40ms
QRS Interval	100ms	±20ms
PR Interval	160 ms	±40ms

And based on the given table, one can find the time intervals in milliseconds and by comparing it with table 3 [32-35] we can find whether ECG waves are normal or abnormal. The normal beats per minute (bpm) is 60 to 100 bpm. Further, the windowing algorithm [36] is used to detect different intervals. R peaks are prominent peaks in the ECG signal, and here they are the highest values in the cycle. After detecting different intervals and comparing it with table 3, the wave is normal or abnormal is discovered. The algorithm written below is the computational task, which is to be done by every node for every single wave. And from the assigned set of waves, if any abnormal wave is discovered then the slave node will do the specified action and it will report to the master dispy node.

Algorithm 1 Windowing Algorithm

Input: ECG wave, **Output:** Ƞ

// Ƞ ∈ {normal, abnormal}

// i → each ECG wave ranging from P-T-R

1. **procedure** detect
2. **for** i← 1 to n **do**
3. R-R interval is $t_{rr} = \frac{R_{(i+1)} - R_{(i)}}{f_s}$
4. P-R interval is $t_{pr} = \frac{R_{(i)} - P_{(i)}}{f_s}$
5. QRS interval is $t_{qrs} = \frac{(S_{(i)}+\theta) - (Q_{(i)}-\theta)}{f_s}$
6. QT interval is $t_{qt} = \frac{T_{(i)} + (t_{rr} * 0.13) - (Q_{(i)}-\theta)}{f_s}$
7. bpm= $\frac{t_{rr} * 60}{f_s}$
8. **end for**
9. **end procedure**

5. PROCESSING ECG SIGNAL USING A DISPY MANNER

The series of ECG waves are divided into a set of two waves to find the ECG intervals. From these ECG intervals, the prediction for the wave is normal or abnormal is made. Here, each job given by master dispy node to the slaves containing two ECG waves and the process continues for a different number of waves. The time taken by different nodes for a different number of waves is as shown below in figure 2.

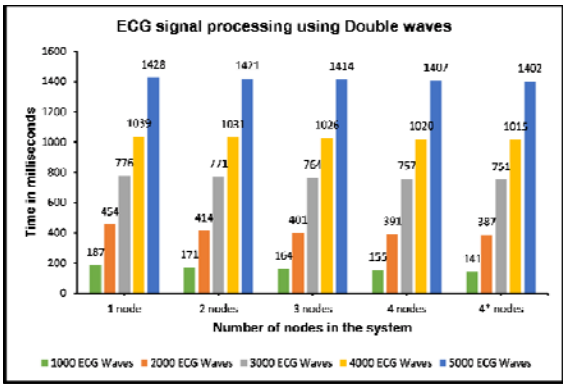

Figure 2: ECG signal processing using two ECG waves in one sub job

Journal of Theoretical and Applied Information Technology
30th November 2020. Vol.98. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

For the system set up mentioned and is used to get figure 2 results, the dispy master node have to do so many iterations to complete the job.

Here every iteration involves communication overhead, socket communication, and task distribution work. Here the average ECG wave size is of 1392 bytes. In any communication system network performance [37] is measured by bits per second. In Layer-2, the minimum frame size is 84 bytes and the maximum frame size is of 1538 bytes. The payload every frame can carry are 46 bytes and 1500 bytes. So if we allow maximum number of waves then it will allow larger frames and hence by reducing the number of frames. This results in lesser overhead of 38 bytes header per frame. So to reduce the number of iterations, the different number of waves are taken in a single job. Figure 3 shows the processing time details for different nodes when each sub-job of the Slave node is carrying the four ECG waves.

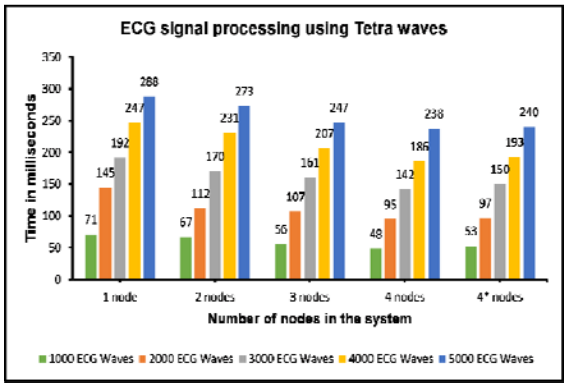

Figure 3: ECG signal processing using four ECG waves in one sub job

When four waves are used in sub-jobs, it is observed that the dispy system computes the task in lesser time. This is due to fewer iterations which are causing fewer delays and overheads. To check further and to reduce processing time, the eight ECG waves are used. In this grouping, every sub job dispatched by master-dispy to the dispy slaves is having eight sets of sequential ECG waves from the main job.

The time taken to process all waves is shown in figure 4. When the set of eight waves are used, the system performance is noticeably improved. It is reducing the processing time very much and it is due to fewer overheads caused in the dispy system.

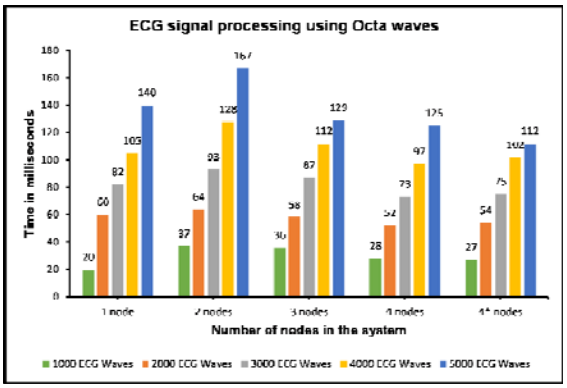

Figure 4: ECG signal processing using eight ECG waves in one sub job

To reduce it further the set of ten ECG waves in one sub-job is tested. The performance measure for the same is shown in figure 5.

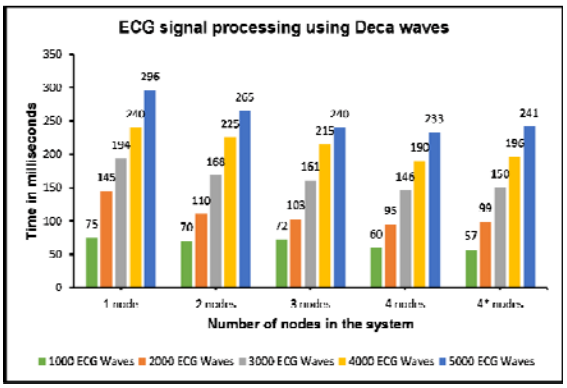

Figure 5: ECG signal processing using ten ECG waves in one sub job

The performance observed in ten wave scenario is not better than the octa wave scenario. This is because the cluster nodes are preloaded with computation and they are receiving ten waves for the processing in one job. These ten waves carry more data to handle and more processing to do, which is causing more computations on the pre-loaded nodes and hence they are taking more computation time for each sub job. So in the case of deca waves, the task distribution and communication overhead are overall lower than the octa wave case, but the computation time is more. This is the reason that the deca wave scenario is not performing as expected. Hence further in this paper, the octa wave as a sub job instance is taken as a reference and further improvements are performed on it.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

6. SYSTEM SYMBOLS, FUNCTIONS AND PERFORMANCE INDICATORS AND THEIR IMPLICATIONS

The different functions, terms, and symbols defined in the proposed system are listed below.

6.1 System parameters

Table 4: System symbols and description

Symbol	Description
J	Complete available job at the master node
J_i	Sub job of J, $J_i \subseteq \{J\}$: sub job with i ECG waves
M	Master Node
S	Slave Node
S_i	i^{th} Slave Node
n	Number of available slave nodes in the dispy system
X_i	X^{th} factor for i^{th} slave node
s	Sub job in sending context
r	Sub job in receiving context
T_s	Sending time of a particular sub-job on the master node
T_r	Receiving time of a particular sub-job on the master node
T_{is}	Sub job sending time to i^{th} node from master node
T_{ir}	Sub job receiving time of i^{th} node on the master node
T_{ijk}	T is timestamp value, where i: is slave node number, j: is the sub-job number, k ϵ {s,r}
N	Total number of sub-jobs
J_{ci}	Result of completed sub-job by i^{th} node
$SelectNode_i$	i^{th} node is selected for the further set of sub-job processing
RT	Response time
Ƞ	Ƞ ϵ {normal, abnormal}, i.e., the characteristic of ECG wave
CU	CPU usage
MU	Memory usage
NC	Average of available core % on the node
∀	For all
RT_o	sum of all RT_i , i∀ {1,2,…n}
MU_o	sum of all MU_i , i∀ {1,2,…n}
O_i	string_object having values timestamp, MU_i, CU_i, and NC_i
O_{ijr}	i: is slave number, j: is the sub-job number and r: is receiving context
O_r	Returning object by i^{th} slave
Ᵽ	Priority factor
Ƈ	Capacity factor
Ł	Time factor
µ	Memory factor
ψ	Impact factor
α	Number of jobs in one go, where each job has set of eight ECG waves
α_c	Result of completed α jobs
J_α	Sub jobs with α number of jobs
*	Multiplication
ms	Milliseconds
P_{ACO}	Performance time taken by Response Time technique
P_{CU}	Performance time taken by CPU usage technique
P_{MU}	Performance time taken by memory usage technique
P_{NU}	Performance time taken by a number of cores technique
$P_{OptiFog}$	Performance time taken by OptiFog algorithm
slave_id	Number representing a slave
job_id	Number representing a sub job

6.2 System Functions

Table 5: System functions and description

Functions	Description
timestamp sendTask(sub job, slave_id);	sends sub-job data to a particular slave bearing the mentioned id and returns the timestamp of that event
timestamp receiveTask(result, slave_id);	receive the result of the given task from the slave having the id number and note the timestamp of that event
timestamp getTimestamp( );	returns timestamp
CPUUsageQueryCU(slave_id);	returns CPU usage of slave_id
MUsageQueryMU(slave_id);	returns memory usage of slave_id
NC QueryNC(slave_id);	returns average availability of cores of slave_id
timestamp getFactors(subjob, slave_id);	returns timestamp when giving sub job to slave_id
string_objectreceiveFactors(job_id, slave_id);	returns string_object after processing job_id on slave_id
calculate(Ƈ, Ł, µ);	calculates capacity, time and memory factor

7. FINDING DIFFERENT OPTIMIZATION PARAMETERS AND USING THEM IN THE SYSTEM TO UNDERSTAND ITS EFFECT ON COMPUTATION

To find different parameters and to introduce optimization in the system, different bio-inspired algorithms are studied like Ant Colony Optimization and Genetic Algorithm. These algorithms when compared in context to the existing dispy systems, it is found that Ant Colony Optimization is telling to consider the response time as a performance factor and the Genetic algorithm is indicating to use CPU usage, number of cores and memory.

Journal of Theoretical and Applied Information Technology
30th November 2020. Vol.98. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

7.1 Applying Response time in the proposed System

Before applying the Response time as a system criterion, the response time is understood and its significance is studied [38-39].

A) Understanding Response Time in process execution

In an operating system there are different measures of time:

1. Arrival time (AT) – The time at which the process enters the ready queue

2. Waiting time – time spent by the process inside the ready queue, waiting for execution

3. Response time (RT) – The time between the arrival of the process inside the ready queue and when it is executed/gets into the processor for the first time.

RT = time when process gets first CPU – AT (1)

The time is taken between the submission of a request and the very first response to that particular request. For a good CPU scheduling, algorithm response time should be minimum.

4. Burst time – Time for which the process is under execution in CPU.

5. Completion time – Time at which the process is completely executed.

6. Turn Around Time – The total time required by the process to be executed and it is including the waiting time.

Response Time is important for interactivity but a doubt might arise that the major concern should be how quickly the process is being executed in its entirety rather than the process being served for the first time. But, this idea is fundamentally incompatible since, to finish the process as soon as possible, the job needs to be started quickly and should not be interrupted in between. In the proposed system, the goal is to reduce the computation time of the job by using the Raspberry Pi cluster and the distributed computing approach using Dispy. After understanding the ACO, the target sub-job is given to each and every node by the master node and the completed jobs are taken from the slave nodes. But while giving and taking the jobs, the master node is keeping the records of all the timestamps. And the response time of the node is found out by using the difference in these timestamps. Based on these response times (RT), the better node is chosen for the job and the remaining jobs are given to the node having the best response time.
The algorithm is as follows.

Algorithm 2 Considering the Response Time to process the Job J

Input: Complete job J having continuous ECG waves

Output: Ⅱ for each and every wave

1. **procedure** find_ACO_Node
2. T_{is}←getTimeStamp();
3. **for** i← 1, 2, ……. n **do**
4. T_{ijs}← sendTask(J_i , S_i);
5. T_{ijr}←receiveTask(Jc_i , S_i);
6. RT_i←T_{ijs} - T_{ijr}
7. **end for**
8. $selectNode_i$← min(RT_1, RT_2, ….. RT_n);
10. **for** j←n+1, n+2, ……. N **do**
11. T_{ijs}← sendTask(J_i , $selectNode_i$);
12. T_{ijr}←receiveTask(Jc_i , $selectNode_i$);
13. **end for**
14. T_{ir}←getTimeStamp();
15. P_{ACO} ←T_{ir} - T_{is}
16. **end procedure**

Based on the ACO, a different number of nodes are taken every time and the P_{ACO} is calculated. And it is shown in the graph below.

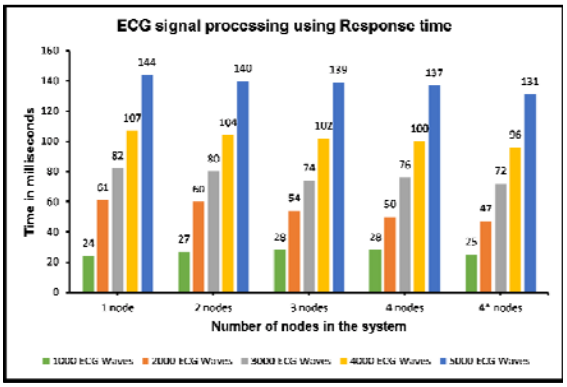

Figure 6: ECG signal processing using response time criteria

Graph Interpretation:

- The performance in 3 nodes and 4 nodes are almost the same. Hence, after a particular time number of nodes does not matter

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

- The effect of 4* node is seen in the performance.
- In 1 Node and 2 Nodes, the performance is not that improved due to the overhead of finding the best node.
- For 4 Nodes, the performance is below the Graph 4 observations, due to the overhead of finding the smallest Response Time.

7.2 ECG signal processing by CPU Usage, Available memory and Number of free cores

Genetic Algorithm says "Survival of Fittest". In the proposed system, the fitness is identified by the different parameters like

- CPU Usage
- Available Runtime memory
- Number of available free cores in the node

All these above three parameters are very essential and they have a major impact on any processing task. To understand the impact of the following parameters in the proposed system, each parameter is taken and the experimentation is performed for the different number of waves and different numbers of nodes.

7.2.1 ECG signal Processing based on CPU Usage

The Central Processing Unit (CPU) performs the Arithmetic, logical and I/O control operations. The fundamental operations of the CPU while executing a process are fetch, decode and execute [40]. These collective operations are known as the instruction cycle. The program counter determines the fetch address and fetches the information from memory. After fetching Program counter jumps to the next instruction present in the sequence. Instruction register stores the fetch address.

In Decode, previously fetched CPU instructions are interrupted to determine CPU's next operation based on the instruction. Every instruction has its unique opcodes, these opcodes are decoded to decide the next operations to be performed.

In Execution, CPU architecture decides to take a single action or a sequence of actions. The intermediate results are stored in the CPU's register for quick access by the very next instructions. Based on the process explained above, the CPU time is calculated as

CPU time = Instruction Count × Clock Cycles/instruction × Clock Cycle time (2)

Furthermore, these basic instructions i.e. fetch, decode and execute can be split into sub-operations based on different CPU architecture to achieve a higher degree of pipelining. Various devices in a computer like (Memory, CPU, I/O and Other) communicate with each other through different buses. Types of buses that are most commonly used in process execution are:

Address bus: Address bus carries the address of the data to be accessed or written from the processor to the memory.

Data bus: Data bus carries the data from the processor to the memory and vice versa.

Control bus: The basic instructions are read/write the data, and these control signals are given by the processor to the memory via the control bus.

Here since the CPU usage is found out every time, the job is given to the node where the availability of CPU is the maximum, i.e., the CPU usage is minimum because CPU usage simply shows how busy the CPU is. And minimum usage will assign the task to the comparatively free node.

Algorithm 3 Considering CPU usage to process the Job J

Input: Complete job J having continuous ECG waves

Output: Ⴗ for each and every wave

1. **procedure** find_CPU_Usage_Node
2. T_{is}←getTimeStamp();
3. **for** i← 1, 2, ……. j **do**
4. **for** i←1, 2, ……. n**do**
5. CU_i←queryCU(S_i);
6. **end for**
7. $selectNode_i$← min(CU_1, CU_2, ….. CU_n);
8. T_{ijs}← sendTask(J_i , $selectNode_i$);
9. T_{ijr}←receiveTask(Jc_i , $selectNode_i$);
10. **end for**
11. T_{ir}←getTimeStamp();
12. P_{CU}←T_{ir} - T_{is}
13. **end procedure**

The performance time based on CPU usage is shown in figure 7.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

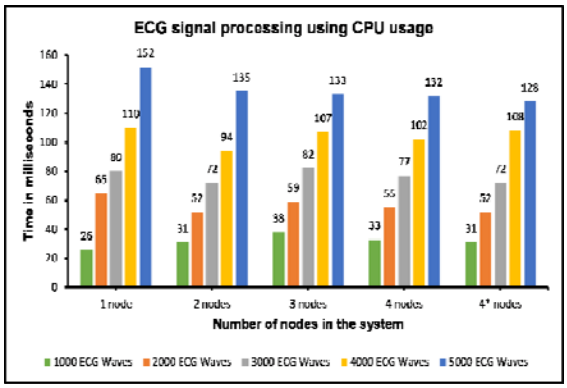

Figure 7: ECG signal processing using CPU usage criteria

Graph interpretation:

- For 1 node the performance is almost the same as the dispy system
- Increase in number of nodes are not helping to improve the performance as it is causing query() to overhead on the system
- Improvement is seen in 4* nodes system

7.2.2 ECG signal Processing based on available main memory

Two broad tasks should be achieved while the operating system manages the computer's memory [41]:

1. Each process must have sufficient memory for its execution, and it cannot overlap into the memory space of another process.
2. The memory in the system should be properly used such that each process can run effectively.

A program under execution is called a process. A program resides in the disk, and it is executed in the main memory. So, it should be transferred from disk to the main memory. From the computation context, a process is defined by its CPU state, memory contents and execution environment. A CPU state is defined by various registers such as Instruction register (IR), Stack Pointer (SP), Program Counter (PC) and general-purpose registers. The memory contains the program code and its predefined data structures. Heap is the reserved memory area for run-time dynamic memory allocation to the program. The stack is used to store the local variables, and the return values of function calls, some register values are also stored in the stack.

- **What does a process look like in memory?**

The process memory is divided into four sections [39]:

- The text section consists of the compiled program code, which reads in from the disk storage when the program is executed.
- In the data section, the static and global variables are stored before executing the main function.
- The heap is used for dynamic memory allocation. This data structure is managed by using calls to new, free, malloc, delete, etc.
- The stack is used for local variables. The stack is reserved for local variables as and when they are declared. When the variables go out of scope, this space becomes free. It is also used to store the return values of the function.

The stack and the heap start at opposite ends of the process's free space and proceeds towards each other. Ideally, they should never meet, if it occurs then either a call to new or malloc will fail, or else a stack overflow error will occur due to insufficient memory. Different processes in the main memory have different address spaces. The memory manager is responsible for managing memory. Programs after completion have to be moved out of the main memory to free the main memory for other processes. Here, the memory usage is the amount of main memory used by the system. For the process, the average memory-access time is calculated as

Average memory_access time = Hit rate + Miss rate × Miss Penalty (3)

The algorithm to process ECG signal using available main memory is as follows.

Algorithm 4 Considering Main Memory to process the Job J

Input: Complete job J having continuous ECG waves

Output: Ŋ for each and every wave

1. **procedure** find_min_MainMemoryUsage_Node
2. T_{is}←getTimeStamp();

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3. **for** i← 1, 2, ……. j **do**
4. **for** i←1, 2, ……. n **do**
5. MU_i←queryMU(S_i);
6. **end for**
7. $selectNode_i$← min(MU_1, MU_2, ….. MU_n);
8. T_{ijs}← sendTask(J_i , $selectNode_i$);
9. T_{ijr}←receiveTask(Jc_i , $selectNode_i$);
10. **end for**
11. T_{ir}←getTimeStamp();
12. P_{MU}←T_{ir} - T_{is}
13. **end procedure**

The memory usage criteria is considered for the system and its performance effect is shown below.

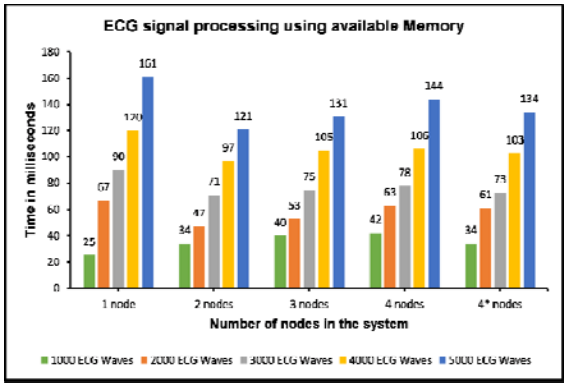

Figure 8: ECG signal processing using available main Memory criteria

Graph Interpretation:

- Increase in number of nodes does not improve the performance
- Less effective than other used techniques
- Asking for available memory is the more overhead for more nodes
- Effect of 4* nodes can be seen

7.2.3 ECG signal Processing based number of cores

A CPU can have more than one processing unit, where each of such units is having independent ALU, registers and control unit [42]. These processing units are called "core". Nowadays, CPUs are available with 8, 10, and higher number of cores. These cores help the CPU, in doing the processing job in a faster manner. As all cores can process simultaneously. There are communication channels available between the cores to communicate the data. The communication channels are in mesh topology form. That is 4 cores will be having six communication channels. In CPU more number of cores can improve the performance [43-44]. Here, every node is queried to find number of cores and their respective usages in percentages. Then every usage is subtracted from 100 to find the availability of the core. Likewise every core availability is added and divided by the number of cores in the node. This is the NC value and the node with highest NC value is chosen and called as node with more available number of cores.

Algorithm 5 Considering the free number of cores to process the Job J

Input: Complete job J having continuous ECG waves

Output: Ƞ for each and every wave

1. **procedure** find_number_of_Free_cores_Node
2. T_{is}←getTimeStamp();
3. **for** i← 1, 2, ……. j **do**
4. **for** i←1, 2, ……. n **do**
5. NC_i←queryNC(S_i);
6. **end for**
7. $selectNode_i$← max(NC_1, NC_2, ….. NC_n);
8. T_{ijs}← sendTask(J_i , $selectNode_i$);
9. T_{ijr}←receiveTask(Jc_i , $selectNode_i$);
10. **end for**
11. T_{ir}←getTimeStamp();
12. P_{NC}←T_{ir} - T_{is}
13. **end procedure**

The Performance measure of this criteria is shown in figure 9.

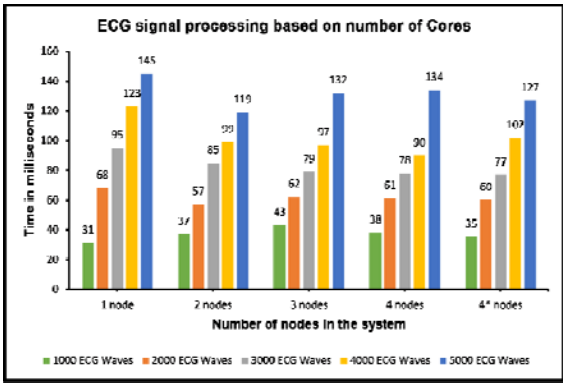

Figure 9: ECG signal processing using a number of Cores criteria

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Graph Interpretation:

- When the number of nodes is more than one, the processing improvement can be seen
- Query overhead can also be seen
- The present of 4* node is helping to improve the performance

8. OPTIFOG ALGORITHM

The proposed idea is intended to perform optimally in the heterogeneous scenario, by exploiting the most available processing power present in the system. We have used four techniques that are memory-based, Response-time-based, CPU-usage-based and number-of-cores-based. Every technique is run when every node was busy in some other computational work. This is just to find out which technique is more weighted and less weighted in heterogeneous computing scenarios. And when the obtained graphs and results are analyzed for greater jobs and higher nodes, it is found that the CPU usage results are the best and the second is the number of cores. Response time gives the third greatest performance followed by memory technique.

8.1 Insights to the OptiFog Algorithm

OptiFog is a hybrid optimization algorithm that finds the impact factor based on all the above four mentioned techniques. This impact factor is the value of every node and based on this value, the number of jobs are allocated to each node in one go. Every node will submit the job and its current status of CPU usage, Cores and Memory to the master node. Master nodes compute the Response time and impact factor for each node in every iteration and based on the impact factor (ψ) value, it will assign the number of jobs.

The CPU and cores on every node have different capacities based on Operating Frequency, processor specifications, cache size and the bus size. It represents the processing capabilities of a node. That is the other main reason to give higher priority (Ᵽ) to this factor. Whereas the memory and response time of a particular node can be compared with other nodes in terms of size and unit like GB and ms. Therefore, these two units memory and response time are seen as collective units in the distributed system.

Impact factor is the overall capability of a node in terms of memory, CPU, cores and response time. But OptiFog uses three main factors to find the impact factor. That is Capacity (Ȼ), Memory (μ) and Time (Ł).

a) Capacity Factor (Ȼ): this factor is based on the CPU usage and number of idle cores technique. The CPU usage is related to the number of cores. And these both techniques are giving a very good performance which is almost similar. So in this case, these two values are combined and the factor is calculated as

$$Ȼ = NC * (1 - CU), \text{where } CU \epsilon [0,1] \quad (4)$$

b) Memory Factor (μ): In this factor, if MU_l is less than the node performs better. Thus, every node MU_l is found out and scaled to 1.The factor is calculated as

$$\mu = \frac{(1-MU_l)}{\sum_{l=1}^{n} MU_l} \quad (5)$$

c) Time Factor (Ł): The response time of a node is inversely proportional to its capacity. By keeping this in mind the factor is designed in such a way that the node with high response time will get low rank and the node with less response time will be treated with high ranks.

$$Ł = \frac{\sum_{l=1}^{n} RT_l}{RT_l} \quad (6)$$

After finding Ȼ, μ and Ł. The final ψis calculated as

$$\psi = 3Ȼ + 2Ł + 1\mu \quad (7)$$

where, numericals ϵ {Ᵽ}

The OptiFog Algorithm is as follows:

Algorithm 6 OptiFog Algorithm to process the Job J

Input: Complete job J having continuous ECG waves,

Output: Ῐ for each and every wave

1. **procedure** OptiFog
2. initialize RT_o =0, MU_o = 0
3. T_s←getTimeStamp();
4. **for** i← 1, 2, ……. n **do**
5. T_{ijs}←getFactors(J_i , S_i);
6. O_{ijr}←receiveFactors(Jc_i , S_i);

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

T_{ijr}← get(O_{ijr});
MU_i← get(O_{ijr});
CU_i← get(O_{ijr});
NC_i← get(O_{ijr});

7. RT_i←T_{ijs} - T_{ijr}
8. RT_o =RT_o + RT_i
9. MU_o =MU_o + MU_i
10. **end for**
11. α_i← 1, ∀ i ϵ [1,2,…n].
12. ψ_l← 0, ∀ i ϵ [1,2,…n].
13. j = n+1, i=1
14. **while j<=N do**
15. calculate(Ҁ$_i$, Ł$_i$, μ$_i$);
16. ψ_{old}←ψ_l
17. $\psi_l = 3Ҁ_l + 2Ł_l + 1\mu_l$
18. **if** ψ_l>ψ_{old} **then**
19. α_i= α_i +1;
20. assignTask(α_i, i);
21. **end if**
22. **if** ψ_l<ψ_{old}**then**
23. α_i= α_i -1;
24. assignTask(α_i, i);
25. **end if**
26. **if** α_l>= 0 **then**
27. j = j + α_i
28. **end if**
29. i = i + 1
30. **if** i > n **then**
31. i = 1
32. **end if**
33. **end while**
34. T_r←getTimeStamp();
35. $P_{OptiFog}$←T_{ir} - T_{is}
36. **end procedure**
37. **Procedure** assignTask(α , i)
38. **if** α_i> 0 **then**
39. T_{is}←getFactors(J$_\alpha$, S$_i$);
40. O_{ir}←receiveFactors(Jαc , S$_i$);

T_{ir}← get(O_{ijr});
MU_i← get(O_{ijr});
CU_i← get(O_{ijr});
NC_i← get(O_{ijr});

RT_i←T_{ir} - T_{is}

41. **end if**
42. **end procedure**

After running the OptiFog Algorithm, the obtained results are shown in figure 10 below.

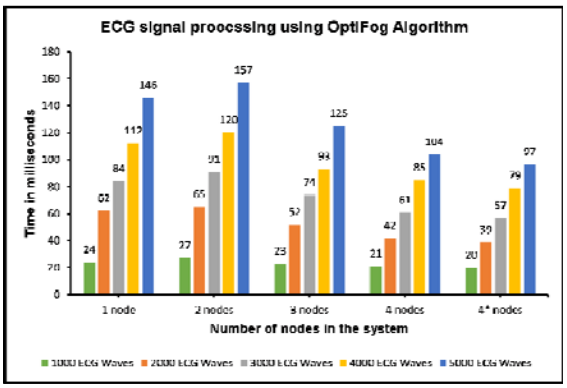

Figure 10: ECG signal processing using OptiFog Algorithm

Graph Interpretation:

- Effect of increasing nodes can be seen
- 4* nodes give very good performance results
- It takes less computation time than ECG octa wave and other techniques

9. TESTING OPTIFOG ALGORITHM

The OptiFog algorithm is tested in three ways to prove its rationality. In the first test case, the speedup factor is considered. In the second test case, the algorithm is run for a higher number of ECG waves to see its performance. And in the last test case the OptiFog algorithm is run for dispy Deca wave case where dispy system which was not giving good performance due to pre-loaded nodes in the cluster.

9.1 Test case 1: Speedup

Based on the above experiments done so far in the Raspberry Pi clustering environment, every parameter or the technique is indicating the way to improvise the performance. These improvements are reducing the job time J, which can be compared by using the Speedup factor to see its impact.

$$Speedup_{overall} = \frac{Execution\ Time_{old}}{Execution\ Time_{new}} \quad (8)$$

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Here, the performance time of the dispy technique with double ECG waves is taken as a benchmark time and other techniques are compared to it. Also, the step by step speedup between other techniques is also shown in Table 6.

Table 6, shows that the series of algorithms are considered are improving the performance. The OptiFog gives the maximum speedup of 14.4536 in 4* Nodes system and it speeds up the performance by 1.1546 with respect to Octa waves in 4* Nodes system.

Table 6: Speedup factors for 5000 ECG waves for 4 nodes and 4 Nodes system*

Technique used (q)	Speedup w.r.t Benchmark		Speedup w.r.t (q-1) technique	
	4 Nodes system	4* Nodes system	4 Nodes system	4* Nodes system
Double waves	1	1	-	-
Tetra Waves	5.9118	5.8417	5.9118	5.8417
Octa Waves	11.256	12.5178	1.904	2.1428
OptiFog Algo.	13.529	14.4536	1.2019	1.1546

9.2 Test Case 2

The proposed algorithm OptiFog is tested on a greater number of ECG waves. The system is tested against the normal dispy system with octa waves and OptiFog algorithm. The system is kept under a loaded scenario to see the performance of the OptiFog algorithm under the worst-case scenario. The number of ECG waves are taken as 5000, 7500 and 10000. The results obtained by the dispy system and OptiFog algorithm is shown in figure 11 and 12.

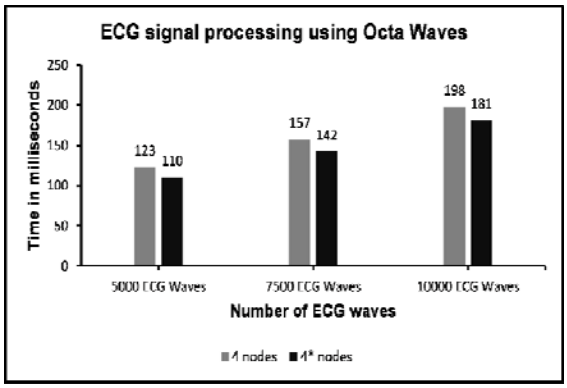

Figure 11: ECG signal processing by dispy system

From Figures 11 and 12, it is observed that OptiFog outperforms the dispy system and other algorithms in terms of computation. OptiFog shows better and better results for larger jobs. In this test case, the effect of 4* nodes in terms of performance is very noticeable. For 4* Nodes system OptiFog is showing a speedup of 1.183 for 10000 ECG waves. This factor was 1.1546 for 5000 waves.

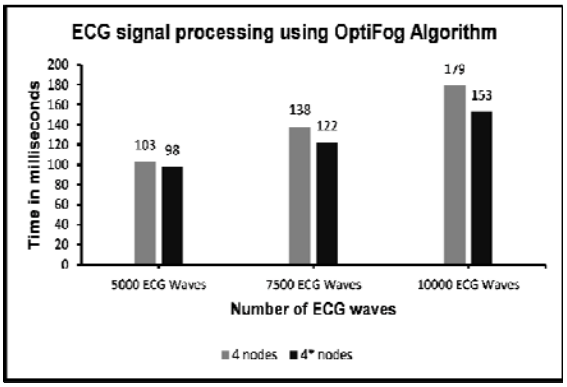

Figure 12: ECG signal processing by OptiFog Algorithm

This confirms that the Speedup factor is improved for higher number of waves. The overall percentage of improvement is shown in figure 13. As the health care processing load increases the OptiFog performs better and its computational performance is increases concerning dispy system.

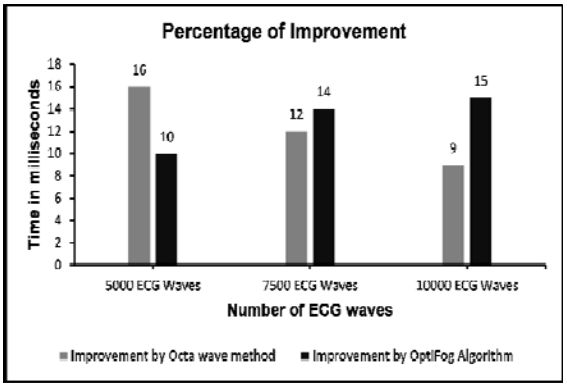

Figure 13: Percentage of improvement in the given test case

9.3 Test case 3

In this test case, the system where dispy was processing ECG signal using deca waves is considered. And when doing this the performance was dropping. In this case, the network overhead is observed lesser because a number of waves are deca, but the loaded nodes are not able to handle the deca wave loads and they are taking more time than expected. But when the same case is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

considered using OptiFog Algorithm which predicts the job size based on individual node capability status using impact ψ value. The result of test case 3 is shown below where OptiFog algorithm is run using deca waves.

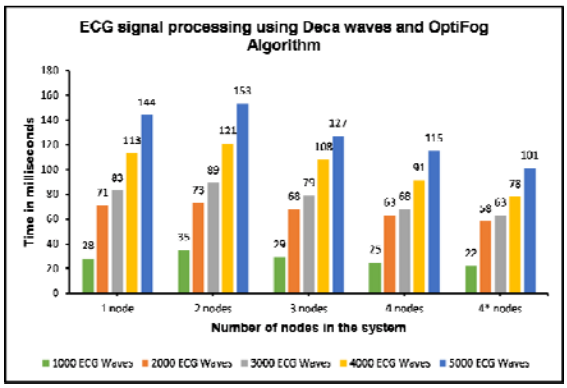

Figure 14: Running Deca waves using OptiFog Algorithm

Graph Interpretation:

- No improvement is seen for 1 node system in comparison to the dispy deca wave graph because of job load and existing preload
- In 2 Nodes system, the performance is degrading because of the calculation overhead of Ć, Ł, μ and ψ.
- For 3 Nodes system, the results are better than 1 Node and 2 Nodes system as now Job assigning and job size variation starts for available nodes
- 4 Nodes system performs better than 1 Node, 2 Nodes and 3 Nodes system
- Finally, 4* Nodes system outperforms because OptiFog is able to detect good impact factors every time and able to assign more and more task in sub job for 4* system

OptiFog uses Distributed computing to strengthen itself, and it is real-time processing algorithm expected in [45]. The Use of Heterogeneous computing and considering the worst case scenario makes it different from the work done, so far in this field.

10. CONCLUSION AND DISCUSSION

Real-time ECG analysis is a time-sensitive health care application. Delay in this can be life-threatening for patients. Fog Computing is able to do this with reduced transmission delays but it lacks in the computation power [46]. To get reduce computational delay Raspberry Pi cluster is suggested. Dispy is a good tool to use in Pi cluster to facilitate ease of deployment and scalability in distributed computing. To get good performance from the dispy system, the assigned sub-job size should be optimal. The master node iterations and overheads depend on the sub-job size, which can affect the system performance at greater levels. Every hardware and software parameter matters a lot in terms of computation. In this system, four parameters namely response time, CPU usage, number of cores and memory is used. Each parameter has its effect on computation. For the current system, the CPU usage and number of free cores were having a good impact while response time and memory had less impact on system performance. By considering these effects and their level of impact, OptiFog algorithm is designed with respect to different priorities and factors. The impact factor is a good measure to determine the processing capability of any node. OptiFog algorithm performs fairly well for the ECG health care data using a Raspberry Pi cluster. OptiFog algorithm is designed for the worst-case scenarios so that it should always perform better. In a heterogeneous environment it is able to decide and assign the optimal job size for different nodes. Shown test cases are justifying the performance of the OptiFog algorithm as, if the number of nodes and job increases, then the algorithm performance will also gradually increase in comparison to dispy systems. Hence OptiFog algorithm is able to achieve better computations in the Heterogeneous Raspberry Pi clustering environment in Fog Computing. In the future, different computing parameters like cache size, clock speed, communication bus size, and processor type can be considered to make this algorithm better and better. Applying and analyzing OptiFog algorithm in Vehicular Fog Computing for different application domain is the open research issue.

REFRENCES:

[1] Bonomi, F., Milito, R., Zhu, J., and Addepalli S, "Fog Computing and its role in the internet of things", *ACM Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp.13-16, 2012

[2] Shi, Y., Ding, G., Wang, H., Roman, H. E., and Lu, S, "The Fog Computing service for healthcare", *IEEE 2nd International Symposium on Future Information and Communication Technologies for Ubiquitous HealthCare (Ubi-HealthTech)*, pp.1-5, 2015.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[3] Bharathi P.D., Ananthanarayanan V., Bagavathi Sivakumar P. (2020) Fog Computing-Based Environmental Monitoring Using Nordic Thingy: 52 and Raspberry Pi. In: Somani A., Shekhawat R., Mundra A., Srivastava S., Verma V. (eds) Smart Systems and IoT: Innovations in Computing. *Smart Innovation, Systems and Technologies*, vol 141. Springer, Singapore

[4] Gareth Mitchell. "The Raspberry Pi single-board computer will revolutionise computer science teaching [For & Against]". *In: Engineering & Technology* 7.3 (2012), 26-26.

[5] Andrew K Dennis. Raspberry Pi home automation with Arduino. Packt Publishing Ltd, 2015.

[6] CG Raji et al. "Implementation of Bitcoin Mining using Raspberry Pi". In: *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT). IEEE*. 2019, pp. 1087-1092.

[7] Suzanne J Matthews et al. "Portable parallel computing with the raspberry pi". *In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 2018, pp. 92-97.

[8] C. Pahl et al. "A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters". In: *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (Fi-CloudW). 2016*, 117-124.

[9] P. Jutadhamakorn et al. "A scalable and low-cost MQTT broker clustering system". *In: 2017 2nd International Conference on Information Technology (INCIT)*. 2017, pp. 1-5.

[10] Pekka Abrahamsson et al. "Affordable and energy-efficient cloud computing clusters: The bolzano raspberry pi cloud cluster experiment". In: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. vol. 2. IEEE. 2013, pp. 170-175.

[11] D. Borthakur et al. "Smart fog: Fog Computing framework for unsupervised clustering analytics in wearable Internet of Things". *In: 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. 2017, pp. 472-476.

[12] Richard Brown et al. "Teaching Parallel and Distributed Computing with MPI on Raspberry Pi Clusters: (Abstract Only)". *In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education. SIGCSE '18. Baltimore, Maryland, USA: Association for Computing Machinery*, 2018, p. 1054. ISBN: 9781450351034.

[13] G. L. Stavrinides and H. D. Karatza, "Task Group Scheduling in Distributed Systems," *2018 International Conference on Computer, Information and Telecommunication Systems (CITS)*, Colmar, 2018, pp. 1-5.

[14] A. Guermouche and J. - L'Excellent, "Memory-based scheduling for a parallel multifrontal solver," *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings*, Santa Fe, NM, USA, 2004, pp. 71

[15] Xiaodong Zhang, Yanxia Qu and Li Xiao, "Improving distributed workload performance by sharing both CPU and memory resources," *Proceedings 20th IEEE International Conference on Distributed Computing Systems*, Taipei, Taiwan, 2000, pp. 233-241.

[16] L. Shi, Y. Sun and L. Wei, "Effect of Scheduling Discipline on CPU-MEM Load Sharing System," *Sixth International Conference on Grid and Cooperative Computing (GCC 2007)*, Los Alamitos, CA, 2007, pp. 242-249.

[17] Kizhakkethil, Sree and S., Murugan. (2017). Memory based Hybrid Dragonfly Algorithm for Numerical Optimization Problems. *Expert Systems with Applications.* 83. 10.1016/j.eswa.2017.04.033.

[18] Mohammad I. Daoud and Nawwaf Kharma, " A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks", *Journal of Parallel and Distributed Computing*, Volume 71, Issue 11, November 2011, Pages 1518-1531.

[19] H. Topcuoglu, S. Hariri and Min-You Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *in IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, March 2002.

[20] Dongning Liang, Pei-Jung Ho, Bao Liu. Scheduling in Distributed Systems. https://cseweb.ucsd.edu/classes/sp99/cse221/projects/Scheduling.pdf

[21] Arash Ghorbannia Delavar,Mahdi Javanmard , Mehrdad Barzegar Shabestari and Marjan Khosravi Talebi, "RSDC (RELIABLE SCHEDULING DISTRIBUTED IN CLOUD COMPUTING)", *International Journal of Computer Science, Engineering and Applications (IJCSEA)* Vol.2, No.3, June 2012.

[22] Luiz F. Bittencourt, Alfredo Goldman, Edmundo R.M. Madeira, Nelson L.S. da Fonseca, Rizos Sakellariou, "Scheduling in distributed systems: *A cloud computing*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

perspective", Computer Science Review 30 (2018) 31–54.

[23] Zafeirios C Papazachos, Helen D Karatza, "Gang scheduling in multi-core clusters implementing migrations", *Future Generation Computer Systems*, Vol. 27, No. 8.

[24] Salim Bitam, SheraliZeadally and AbdelhamidMellouk (2018) Fog Computing job scheduling optimization based on bees swarm, *Enterprise Information Systems*, 12:4, 373-397.

[25] F. A. Kraemer, A. E. Braten, N. Tamkittikhun and D. Palma, "Fog Computing in Healthcare–A Review and Discussion," *in IEEE Access*, vol. 5, pp. 9206-9222, 2017.

[26] D. R. Ries and G. C. Smith, "Nested Transactions in Distributed Systems," *in IEEE Transactions on Software Engineering*, vol. SE-8, no. 3, pp. 167-172, May 1982.

[27] dispy: Distributed and Parallel Computing with/for Python by GiridharPemmasani, https://pgiri.github.io/dispy/

[28] Raspberry Pi 3 Model B+, https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[29] Raspberry Pi 4 Model-B with 4 GB RAM, https://robu.in/product/raspberry-pi-4-model-b-with-4-gb-ram/

[30] How to Impose High CPU Load and Stress Test on Linux Using 'Stress-ng' Tool, https://www.tecmint.com/linux-cpu-load-stress-test-with-stress-ng-tool/

[31] Kanani P., Padole M. (2018) Recognizing Real Time ECG Anomalies Using Arduino, AD8232 and Java. In: Singh M., Gupta P., Tyagi V., Flusser J., Ören T. (eds) Advances in Computing and Data Sciences. *ICACDS 2018. Communications in Computer and Information Science*, vol 905. Springer, Singapore

[32] Cardiology Teaching Package. http://www.nottingham.ac.uk/nursing/practice/resources/cardiology/function/normal_duration.php

[33] Standard range of intervals, June 2017. E MEDICINE.http://emedicine.medscape.com/article/2172196-overview

[34] Normal ECG. https://meds.queensu.ca/central/assets/modules/ECG/normal_ecg.html

[35] Eduardo Jose da S. Luz et al., "ECG-based heartbeat classification for arrhythmia detection: A survey", *Computer Methods and Programs in Biomedicine*, Volume 127, April 2016, Pages 144-164.

[36] Umer, Muhammad & Bhatti, Bilal & Tariq, Muhammad & Zia-ul-Hassan, Muhammad & Khan, Muhammad & Zaidi, Tahir. (2014). Electrocardiogram Feature Extraction and Pattern Recognition Using a Novel Windowing Algorithm. *Advances in Bioscience and Biotechnology*. 05. 886-894.

[37] Bandwidth, Packets Per Second, and Other Network Performance Metrics, https://tools.cisco.com/security/center/resources/network_performance_metrics

[38] CPU Scheduling, https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html

[39] Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, "Operating System Concepts, Eighth Edition ", Wiley.

[40] Patterson, David A.; Hennessy, John L.; Larus, James R. (1999). Computer Organization and Design: the Hardware/Software Interface (2. ed., 3rd print. ed.). San Francisco: Kaufmann. pp. 751. ISBN 978-1558604285.

[41] How Operating Systems Work, https://computer.howstuffworks.com/operating-system6.htm

[42] CPU and memory, https://www.bbc.co.uk/bitesize/guides/zmb9mp3/revision/2

[43] Monika Mukul and JyotiBala, " STUDY OF MULTI CORE PROCESSOR AND IT'S PERFORMANCE EVALUATION", *IEEE Student Conference on Cognizance of Applied Engineering & Research,* ICAER'10, UIET, Panjab University, Chandigarh.

[44] Pandey, Raksha and Badal, Neelendra, Understanding the Role of Parallel Programming in Multi-core Processor Based Systems (March 11, 2019). *Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE) 2019*. Available at SSRN: https://ssrn.com/abstract=3350311.

[45] S. Sarkar, S. Chatterjee and S. Misra, "Assessment of the Suitability of Fog Computing in the Context of Internet of Things," *in IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 46-59, Jan.-March 2018.

[46] Pratik Kanani and Mamta Padole, "Implementing and Analyzing Health as a Service in Fog and Cloud Computing", The International Journal of Intelligent Engineering and Systems, Vol. 13, No. 6, 2020. DOI: 10.22266/ijies2020.1213.13