International Journal of Computer Networks and Applications (IJCNA)
DOI: 10.22247/ijcna/2021/210720 Volume 8, Issue 6, November – December (2021)

RESEARCH ARTICLE

Optimization of Smart Mobile Device Work Time Using an Optimal Decision Tree Classifier and Data Caching Technique in on Premise Network

Sridhar S K
Department of Computer Science and Engineering, Ballari Institute of Technology and Management, Ballari, Karnataka, India
sridhar@bitm.edu.in

J. Amutharaj
Department of Information Science and Engineering, RajaRajeswari College of Engineering, Bengaluru, Karnataka, India
amutharajj@yahoo.com

Received: 12 September 2021 / Revised: 05 October 2021 / Accepted: 26 October 2021 / Published: 30 December 2021

Abstract – Today, Most smart mobile devices are facilitated with advanced processing hardware and short-range data communication systems by which they are practically capable to provide effective execution services to the neighbor mobile device client request and/or receive services on a need basis within the local area network. Therefore, to relish the powerful capability of these smart mobile devices in the private campus network, we propose an intelligent composite offload decision algorithm (ICODA) framework that attempts to connect several smart mobile devices in wireless local area network and make them apply intelligence before servicing each other request preferably without the internet. The significance of the proposed framework is that it has a mechanism to make a data offloading decision using an optimal decision tree classifier model and also a mechanism to avoid data offloading operation using the data cache neural networks model. The experimental results obtained are obvious to show the minimal client system battery utilization and hence an optimized work time for a smart mobile client device that participates in the ICODA framework.

Index Terms – Private Network, Client-Side Local Cache, Device Status Report Generation, Data Offload Decision, Server Side Global Cache, Average Battery Energy and Task Run Time Measure, Optimized Work Time.

1. INTRODUCTION

The brisk advancement of data communication and machine learning solutions in smart mobile devices has facilitated the way to discover the research knowledge gaps in the on premise computing framework. To optimize the work time and/or battery life consumption, the computational tasks that arise from mobile devices may be executed locally or offloaded to neighboring devices [1]. The data caching mechanism make use of deep learning to predict popular cache data inclusion accurately [2]. The productive artificial intelligence solution with past experiences, powerful models, and an elegant learning ability can robotise the mobile client data offload decision making capability in the network [3]. To overcome the problems of high latency, more energy consumption, high bandwidth, and lack of network connectivity infrastructures, it is sensible to perform local computations on the mobile devices alone [4]. The local mobile devices can release the weight of the workload and reduce the computation costs in local task execution by maintaining a coordinated relationship between mobile devices and servers in offloading frameworks [5]. If offloading frameworks are a well-planned design then the task offloading will optimize the mobile device's computational efficiency with reduced latency and less energy consumption [6]. The combination of optimized, unsupervised and deep machine learning solutions are used to cache data at fog computation model act as a booster to data access in quick time [7]. Computational intensive applications can receive major benefits from data offloading mechanisms than the data-intensive applications which need to spend more time solely on data communication rather computation [8]. “The nearby mobile devices can efficiently be utilized as a crowd-powered resource cloud to complement the remote clouds. The issues related node heterogeneity, unknown worker capability, and dynamism are identified as essential challenges to be addressed when scheduling work among nearby mobile devices” [9]. The proposed intelligent composite offload decision algorithm (ICODA) framework is specially designed for the smart mobile devices attached to a wireless local area network (WLAN) to address their work time problem. The main objectives of real-time implementation of ICODA framework are to create an on

RESEARCH ARTICLE

premise WLAN with the finite number of mobile devices as depicted in Figure 1, develop an intelligent data caching mechanism at both client and centralized server end, and an optimal supervised decision tree classification approach to make the appropriate data offload decision and hence perform remote execution with direct or indirect result transfers based on idle neighbor mobile device server selection and its current wireless network coverage information.

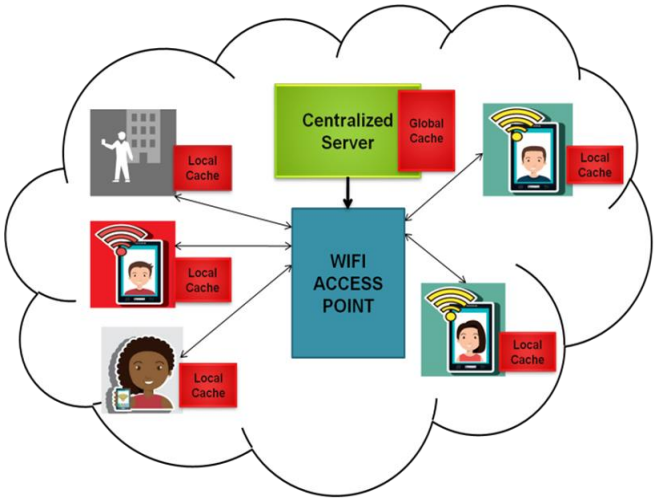

Figure 1 On-Premise Computing Model in ICODA Framework

2. RELATED WORK

A detailed literature review of several different approaches identified for Mobile Data Offloading strategies based on Machine Learning, Task scheduling, Congestion awareness, Server Specification, User Mobility, Mobile Client Energy Estimation, Communication Path Selection, Cache Management, and Application workload structure has been presented in Table 1. The JAY model can performs computation on local device network based on runtime system profiler information with fast and privileged access to raw data [1]. The OREGANO model performs computation in batches and data streams on private mobile network where the data reside thereby reducing the data communication overhead associated with cloud offload computing [4]. The compute intensive files can gain more benefits out of data offloading techniques compared to data intensive high size files [8].

The mobile device heterogeneity, unknown capability and run time computation are the challenges with data offloading techniques in private local mobile device networks [9]. A proficient cache enhancement method is used to store more popular results in cache server to optimize offloading operation. As the task run time length increases, more important processed information are cached to keep them in tact in the storage server. So, the users intend to move the data to the server to reduce the run time latency. On simulation it is observed that "it reduces execution delay up to 42:83% and 33:28% for single-user femto-cloud and single-user mobile edge computing, respectively. Also for multi-user OOCS can further reduce 11:71% delay compared to single-user OOCS through user's cooperation" [10].

The automated selection of off-loadable code using @offload annotations, divides the prime task into off-loadable and non-off-loadable components. It is the application developer with expertise, who inserts the annotations for selected methods that can preferably get benefited by data offloading mechanism. The simulation resulted in less run time and battery consumption value [11]. An experiment has been conducted based on file size with different wireless mobile networks using MECCA model. Different task size and different network interfaces are made part of implementation to measure energy consumption and computation time and then perform the required comparative analysis. "The results obtained have revealed the cloud potential in the reduction of power consumption by 61% to 90% for Wi-Fi and 4G respectively" [12].

The energy-oriented task scheduling and weight assignment scheme uses energy status and local computing power to guarantee low residual energy mobile client devices to get scheduled first for computation which leads more user satisfaction in network [13]. The architecture of mobile edge cloud computing has been designed as a computation offload strategy for mobile devices. It incorporates a deep learning solution to predict task size, required CPU cycles and total transfer delay to make appropriate data offload decision and a computation task migration algorithm for edge cloud on failures. It divides an ongoing task into several small tasks and each small task is executed on a node [14]. The adaptive task allocation approach uses energy consumption estimator based on power profiles that can increase the execution performance during data offload operation.

The experiments are conducted for local device, Wi-Fi and 3G transmissions using machine solution for transfer of source code [15]. The Adaptive job allocation scheduler (AJAS) allocates jobs to slave nodes using on the go computing resources and current battery status. The AJAS job processing time is relatively faster than dynamic, static and random based job allocation methods [16]. The computation offloading approach constitutes of a Deep Belief Network (DBN) and a logistic regression layer. The typical binary RBM is changed to a Gaussian–Bernoulli RBM Learning technique. It learns from the request and response record of nodes in local network and then the response time of subsequent requests are predicted [17]. The necessity of an intelligent offloading framework for work time optimization is identified as a research gap.

RESEARCH ARTICLE

Ref. No.	Offload Decision Criteria	Methodology / Algorithm Used	Limitation / Gap Identified
[1]	Based on local applications and system profiler	JAY- Runtime system profile generation task scheduling algorithm	A Configurable cloud. No data caching. Excess profiled data communication.
[4]	Based on data location, the computations are performed on the network where the data reside.	Data-centric mobile computing aware processing of tasks generated by co-located mobile devices (OREGANO model).	No data caching. Device battery and CPU power are not considered for load balancing operation.
[10]	Most important computation results are stored in the cache server.	Optimal Offload with Cache Enhancement Scheme(OOCS)	Significant reduction. Just Simulated.
[11]	Based on "@offload" annotation in code by developers.	User Level Online Offloading Framework.	User Mobility, Multi-User & Server not considered.
[12]	Based on different task size input to 3G, 4G & Wi-Fi network interfaces.	MECCA Rule-based approach	Mobile Client-oriented.
[13]	Low residual Energy mobile clients are preferred first.	Energy and priority task-oriented scheduling scheme.	Local energy and computation status-oriented
[14]	The necessary CPU cycles, maximum uplink, and data communication time are computed with proper workload size prediction.	Deep Learning solution Sub Task Migration Algorithm.	No Content Caching.
[15]	Adaptable Task Allocation. Energy consumption estimator.	Decision Tree. K-Neural Networks. Power profiler application programming interface.	Regression, 4G/5G.
[16]	Based on Mobile device battery level	Master slave concept Dynamic computer resources allocation.	More focus on job rescheduling. Just Simulated.
[17]	Based on request & response record of nodes, It predicts response time of subsequent request.	Deep Belief Network with Regression Layer.	Just Simulated. No Real-world scenario.
[18]	Local Mobile Cloud Energy Sharing Effect	ColloboRoid Architecture	Applies only to Mobiles in Same Access Point.

Table 1 Literature Review on Different Offloading Strategies

3. THE PROPOSED ICODA FRAMEWORK

The proposed ICODA framework consists of 4 major components as shown in Figure 2.

3.1. Client-Side Local Cache (CSLC) Component

The working of CSLC begins with the selection of processer bound workload on a mobile client device. It checks the existence of the respective workload transaction history in the local cache. If the local cache entry is found, then the corresponding output file is immediately made available to a client device. Otherwise, it activates the client device status report generation component. The local cache area contains 8 attributes as shown in Table 2, where the first 6 attributes are crucial to evaluate the target variable 'X-factor' value. The last attribute is a corresponding pointer to an output file location.

International Journal of Computer Networks and Applications (IJCNA)
DOI: 10.22247/ijcna/2021/210720 Volume 8, Issue 6, November – December (2021)

RESEARCH ARTICLE

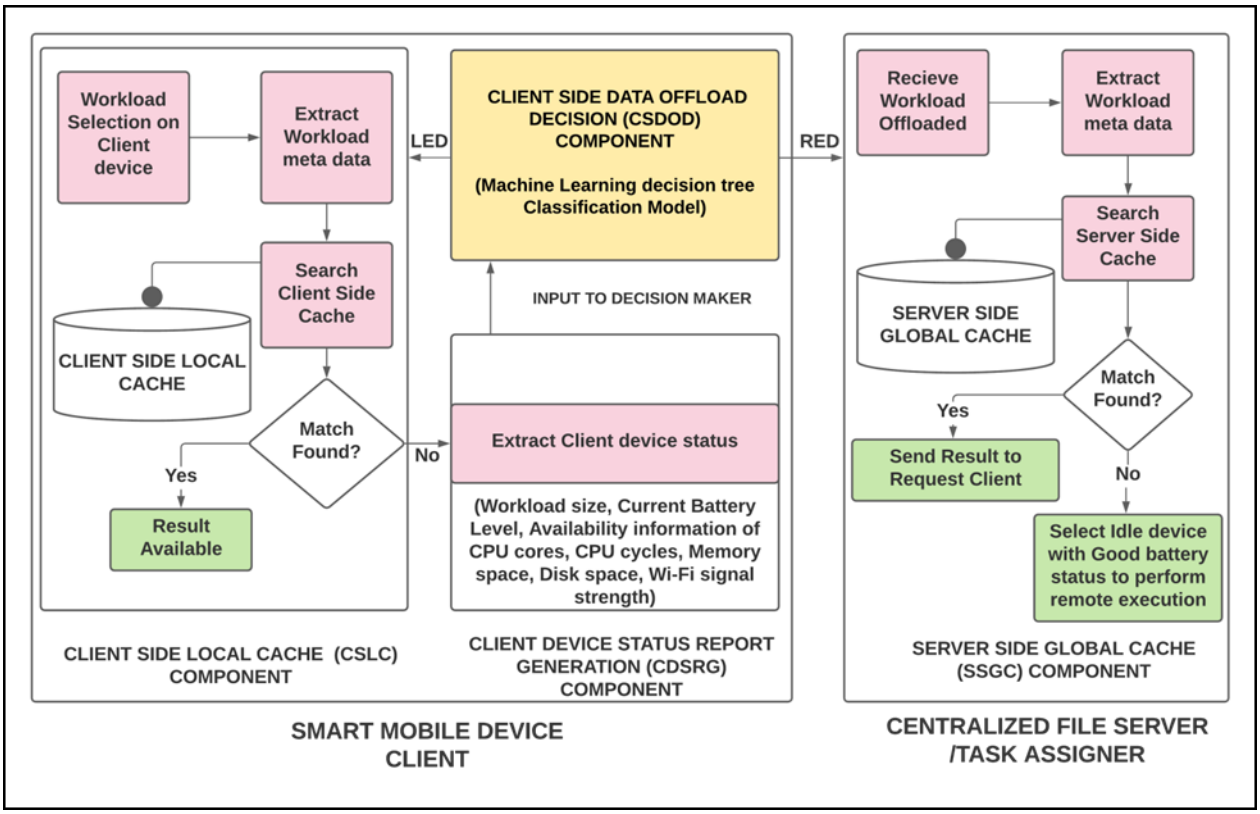

Figure 2 Proposed ICODA Framework

File Name	File Size (Bytes)	Previously Offloaded?	File Selection count	Remote Run Time (Seconds)	Local Run Time (Seconds)	X-factor	Pointer to Output File
P2.py	512	NO	22	0.0	16.49	1.41	Ptr1
P4.py	586	NO	5	0.0	20.34	5.76	Ptr2
P8.py	610	YES	14	26.28	0.0	0.82	Ptr3
L.py	827	NO	8	0.0	33.05	3.12	Ptr4
H.py	1024	YES	10	40.66	0.0	2.51	Ptr5

Table 2 Client-Side Local Cache Attributes

3.1.1. Local Cache Replacement Policy

The cache data replacement policy relies on File size, Offloaded history, File selection count, Remote and Local run time attributes to compute X-factor value as shown in Eq. (1). Based on this X-factor value, a 'victim' row of data replacement is identified as shown in Table 3. The cache data replacement policy uses the X-factor value as major criteria to identify victim.

$$X - factor = \frac{FS}{(FSC \times [(2 \times RRT) + (LRT)]} \quad (1)$$

In Eq. (1), FS, FSC, RRT, and LRT denote File size, File selection count, Remote run time, and Local run time

RESEARCH ARTICLE

respectively. The 'Victim' indicates the file location to be replaced with a new subsequent popular file data with high run time and low X-factor.

Larger the 'X-factor' value, the corresponding file get to be a victim of data replacement. If there is a tie between two rows then the locally executed file gets replaced with new data. The 'Retain' value indicates that there is no replacement currently for the row in a cache. This cache area gets more populated preferably with the source files that are remotely executed with high run time measures. The design flow of client side local cache search and replacement policy are presented in algorithm 1 and algorithm 2.

File Name	File Size (Bytes)	Previously Offloaded?	File Selection count	RRT (Sec)	LRT (Sec)	X-factor	Pointer to Output File	Local cache Replacement
P2.py	512	NO	22	0.0	16.49	1.41	Ptr1	Retain
P4.py	586	NO	5	0.0	20.34	5.76	Ptr2	Victim 1
P8.py	610	YES	14	26.28	0	0.82	Ptr3	Retain
L.py	827	NO	8	0.0	33.05	3.12	Ptr4	Victim 2
H.py	1024	YES	10	40.66	0	2.51	Ptr5	Victim 3

Table 3 Local Cache Data Replacement Policy Based on X-Factor

Algorithm: Client-side-Local-cache-search

Input: Workload name, Workload size.

Output: Display related output file

BEGIN

Check *local cache area* for metadata match of input workload (W).

If *workload entry match*es with entry in Local cache:

Then

Display *corresponding output file.*

Exit

If *No workload entry match*es in the Table:

Then

Invoke *Report Generation* component (CDSRG).

Receive *output file with run time measure*.

Display *output file*.

Invoke Local-cache-insert algorithm.

END

Algorithm 1: Client-Side-Local-Cache-Search

Algorithm: Local-cache-insert

Input: Workload name, Workload size, LRT/RRT.

Output: Transaction Insertion

BEGIN

If *local cache space == "Available"*:

Then

Increment FSC by 1 for the workload (W)

Compute *X-factor value.*

Add transaction to local cache table.

Exit

If *local cache space =="Unavailable"*:

Then

Apply *cache replacement policy.*

Find *cache data entry with high X-factor.*

Replace *it with new transaction data.*

END

Algorithm 2 Local-Cache-Insert

3.2. Client Device Status Report Generation (CDSRG) Component

The CDSRG component collects seven device features from smart mobile client device profiled resource status data such as File size, current battery level, CPU core, CPU cycles utilization percentage, physical memory availability, physical hard disk availability, and Wi-Fi Signal strength to form an input report to submit to data offload decision component. The workflow of CDSRG component is presented in algorithm 3.

Algorithm: Client-device-status-report-generation

RESEARCH ARTICLE

Input: Workload path

Output: Profiled resource consumption information

BEGIN

Obtain Workload Name, Format, and size in bytes to be offloaded.

Obtain present battery status of the client device.

Obtain number of CPU cores in the client device.

Obtain current availability percentage of main memory space in the client device.

Obtain current availability percentage of storage space in the client device

Obtain Network signal strength of connection for communication.

Call Random forest decision tree classifier function in CSDOD component.

END

Algorithm 3 Client-Device-Status-Report-Generation

3.3. Client-Side Data Offload Decision (CSDOD) Component

It uses client device status information report as input to machine learning classification algorithm, embedded in CSDOD component to generate an optimal and accurate data offload decision either to move a computational workload to the centralized server connected within a private network or to perform local execution itself. The comparable investigation has been carried out between Iterative Dichotomiser 3, CART, and Random forest classifier (RFC) with cross-validations. The Random forest decision tree classifier has outperformed the other two with 99.44 classification accuracy. The detail of comparative analysis has been presented in Results and discussion section. The workflow of CSDOD component is presented in algorithm 4.

Algorithm: Random-forest-decision-tree-classifier

Input: Profiled client device resource status report

Output: Local execution decision (LED) / Remote execution decision (RED)

BEGIN

Select random K data instances from a specified training dataset input.

Generate decision trees for each sub-sample of the dataset

Predict the output from each decision tree.

Perform poll for each predicted result.

Output the majority voted prediction result.

END

Algorithm 4 Random-Forest-Decision-Tree-Classifier

3.4. Server-Side Global Cache (SSGC) Component

In this component, the centralized server maintains a database that consists of workload name, size in bytes, run time, and a pointer to a corresponding output file of all computational workloads that have undergone remote execution. Therefore it first performs a global cache search to check the existence of respective workload transaction history. If a transaction is found then the corresponding output file is immediately directed towards to the actual client device. Otherwise, it selects an idle neighbor device with good battery level to perform remote execution. Once the remote execution is completed, it is the responsibility of the centralized server to update the cache with a new workload entry consisting of its name, size in bytes, run time, and a corresponding pointer to the output file in the server cache and send the same information to the actual client to update itself. Hence, the overall mobile client device work time can be optimized by reducing the burden of processing workload on a client device. The device work time is the period for which the mobile device is operable to perform essential application execution. The global cache area contains 6 attributes as shown in Table 4, where the first 4 attributes are crucial to evaluate the target variable 'X-factor' value. The last attribute is a corresponding pointer to the output file location.

3.4.1. Global Cache Replacement Policy

Whenever the centralized server directs to update the global cache, the global replacement policy gets activated. The global cache data replacement policy relies only on File size (FS), File selection count (FSC) and Remote run time (RRT) attributes to compute X-factor value as shown in Eq. (2). Based on this X-factor value, a 'victim' row of data replacement is identified as shown in Table 5. The formula for X-factor value computation is:

$$X - \text{factor} = \frac{FS}{FSC * RRT} \qquad (2)$$

The global cache is beneficial to avoid unnecessary data offloading operations which incur the cost of device selection, data transmission, and network traffic. The global cache holds the most important and high run time-oriented source files with their respective output files collected from all user mobile participants in the CODA framework and hence can be treated as a more diversified cache data relevant to the specific organization. The workflows of SSGC component in global cache search and data replacement policy are presented in algorithm 5 and 6.

RESEARCH ARTICLE

File Name	File Size (Bytes)	File Selection Count	Remote Run Time (seconds)	X-Factor	Pointer to Output File
PI-2.py	658	12	16.49	3.32	Sptr1
PI-4.py	575	23	20.34	1.22	Sptr2
PI-8.py	756	41	26.28	0.70	Sptr3
CC.py	950	32	33.05	0.89	Sptr4
DC.py	1024	11	300.66	2.28	Sptr5

Table 4 Server-Side Global Cache Attributes

File Name	File Size (Bytes)	File Selection Count	Remote Run Time (Seconds)	X-Factor	Pointer to Output File	Global Cache Replacement
PI-2.py	658	12	16.49	3.32	Sptr1	Victim 1
PI-4.py	575	23	20.34	1.22	Sptr2	Victim 3
PI-8.py	756	41	26.28	0.70	Sptr3	Retain
CC.py	950	32	33.05	0.89	Sptr4	Retain
DC.py	1024	11	300.66	2.28	Sptr5	Victim 2

Table 5 Global Cache Data Replacement Policy Based on X-Factor

Algorithm: Server-side-Global-cache-search

Input: Workload file, Workload name, and its size

Output: Display related output file

BEGIN

Check g*lobal cache area* for metadata match of input workload (W).

If *workload entry match***es** in the Global cache:

Then

Display *corresponding output file.*

Exit

If *No workload entry match*es in the Global cache:

Then

Select an *Idle neighbor device with battery stability* to perform remote execution.

Receive *output file* with run time measure.

Send *output file to an actual request client device.*

Invoke Global-cache-insert algorithm.

END

Algorithm 5 Server-Side-Global-Cache-Search

RESEARCH ARTICLE

Algorithm: Global-cache-insert

Input: Workload name, Workload size, RRT.

Output: Transaction Insertion

BEGIN

If *global cache space == "Available"*:

Then

Increment FSC by 1 for the workload (W)

Compute *X-factor value.*

Add transaction to global cache table.

Exit

If *global cache space =="Unavailable"*:

Then

Apply *cache replacement policy.*

Find *cache data entry with high X-factor.*

Replace *it with new transaction data.*

END

Algorithm 6 Global-Cache-Insert

4. IMPLEMENTATION

The experimental setup comprises of 81 mobile devices which include smartphones and laptops of different specifications. The smart devices are powered by android version operating system, RAM component ranging from 2 to 8 Gigabytes capacity, processor core ranging from dual to octal numbers, internal storage ranging from 64 to 128 GB and battery 4000mAh to 6000mAh. The participated laptop computers are powered by 64-bit Windows 10 operating system with Intel core i3, i5 and i7 operable at 2.2, 2.3 and 2.5 GHz. These laptops has RAM ranging from 4 to 8 GB and battery capacity ranging from 2.29 to 3.3Ah. These specifications of mobile devices are mentioned to inform that all the different mobile devices were involved in the implementation and have no considerable effect on ICODA performance. The laptops are just part of the mobile network which can request/offer services to neighbor mobile devices under the supervision of the centralized server. The Python application programming is used to design and implement an ICODA mobile application framework using several popular graphic, process and system utility packages. The PyCharm integrated development environment with installed Python version 3.9 interpreter is utilized to run the python project. Once the mobile application is developed, it is inputted to a Buildozer tool to translate a kivy application into an android compatible java application to make it run on real android supported mobile devices. The real mobile devices are then connected over Wi-Fi to communicate the data with each other without necessarily using the internet is performed using special network and decision tree classifier packages in python. A 40000 real-time profiled resource consumption data samples are noted in the experiment of 80 mobile devices connected through a wireless access point in the private network, offloading compute-intensive files of size ranging from 500KB to 1 MB among each other through the centralized server at different time conditions. A sample of profiled resource consumption data is shown in Table 6.

Suppose the Client selected workload: D:\\Files\CI-2.pyz		
ATTRIBUTE	**VALUE**	**RANGE**
Workload Size	0.5 MB	0 < size < 1 MB
Client Battery Status	72 %	25 – 100 %
Client CPU Count	4	02 – 04 – 08
Client CPU Cycles %	96.3	50 – 100 %
Client Virtual Memory %	55.8	50 – 100 %
Client Disk Storage	97.5%	50 – 100 %
Wi - Fi Network Signal Strength	Good	-67 to -50 dBm

Table 6 A Real-Time Profiled Resource Consumption Data Sample

The ICODA resource status report contains 7 prime input attributes obtained from smart mobiles devices and a single target variable i.e. data offload decision (DOD). The threshold setting for each attribute is presented in Table 7.

Based on the threshold setting as shown in Table 7, the real-time large data set of 40000 data samples have been normalized to the practice range with several finite categories for each feature. After normalization, the total training samples that represent the large data set is reduced to just 2*4*3*4*4*4*3 = 4608 samples as presented in Table 8. Based on the training, the optimal decision tree classifier accepts 7 input attributes, performs analysis on them, and produces a data offload decision (DOD) on whether to offload the data to a centralized server for further operation or perform local execution. This decision-making intelligence optimizes the overall smart mobile client device work time and reduces the regular battery usage on specific and significant workloads.

RESEARCH ARTICLE

SIZE	CUR_BAT	CPU_CORE	CPU_CYCLES
0 – Small <= 500 KB	0 – LB (0 – 25)%	0 - Dual	0 – L 75% utilized
	1 – MB (26 – 50)%	1 - Quad	1 – M 50% utilized
1 – Large >500 KB <= 1 MB	2 – HB (51 – 75)%	2 - Octa	2 – H 25% utilized
	3 – Max (76 – 100)%		3 – VH <25% utilized
PHY_MEM	**PHY_DISK**	**Wi-Fi Strength**	**Target feature: DOD**
0 – LM (0 to 25)% Memory Available	0 – LS (0 to 25)% Available	0 – Poor signal	0 – Local execution
1 – MM (26 to 50)% Memory Available	1 – MS 26 to 50)% Available	1 – Good signal	
2 – HM (51 to 75)% Memory Available	2 – HS (51 to 75)% Available	2 – Excellent Signal	1 – Remote Execution
3 – VHM (76 to 100)% Memory Available	3 – VHS (76 to 100)% Available		

Table 7 Threshold Setting for the Resource Consumption Status Report

SIZE (MB)	CUR_BAT	CPU_CORE	CPU_CYCLES	PHY_MEM	PHY_DISK	Wi-Fi Signal	DOD
0.5	LB	D	L	LM	LS	E	Y
0.5	MB	Q	M	MM	MS	G	N
0.5	HB	O	H	HM	HS	E	N
0.5	Max	D	VH	VHM	VHS	P	N
1.0	LB	D	L	LM	LS	E	Y
1.0	MB	Q	M	MM	MS	G	Y
1.0	HB	O	H	HM	HS	E	N
1.0	Max	D	VH	VHM	VHS	P	N

Table 8 A Sample Data Frame of Normalized Data Set Containing 4608 Data Samples

RESEARCH ARTICLE

From Table 8, the values LB, MB, HB, and Max denote low battery, medium battery, high battery, and Maximum battery level. The values D, Q, and O denote dual-core, quad-core, and octa-core processors. The values L, M, H, and VH denote low, medium, high, and very high CPU cycles availability. The values LM, MM, HM, and VHM denote low memory, medium memory, high memory, and very high memory availability. The values LS, MS, HS, and VHS denote low storage, medium storage, high storage, and very high storage availability. The values E, G, and P denote Excellent, Good, and Poor Wi-Fi signal strength. The values Y and N for target feature DOD denote 'yes' to Offload file for remote execution and 'no' to perform local execution respectively.

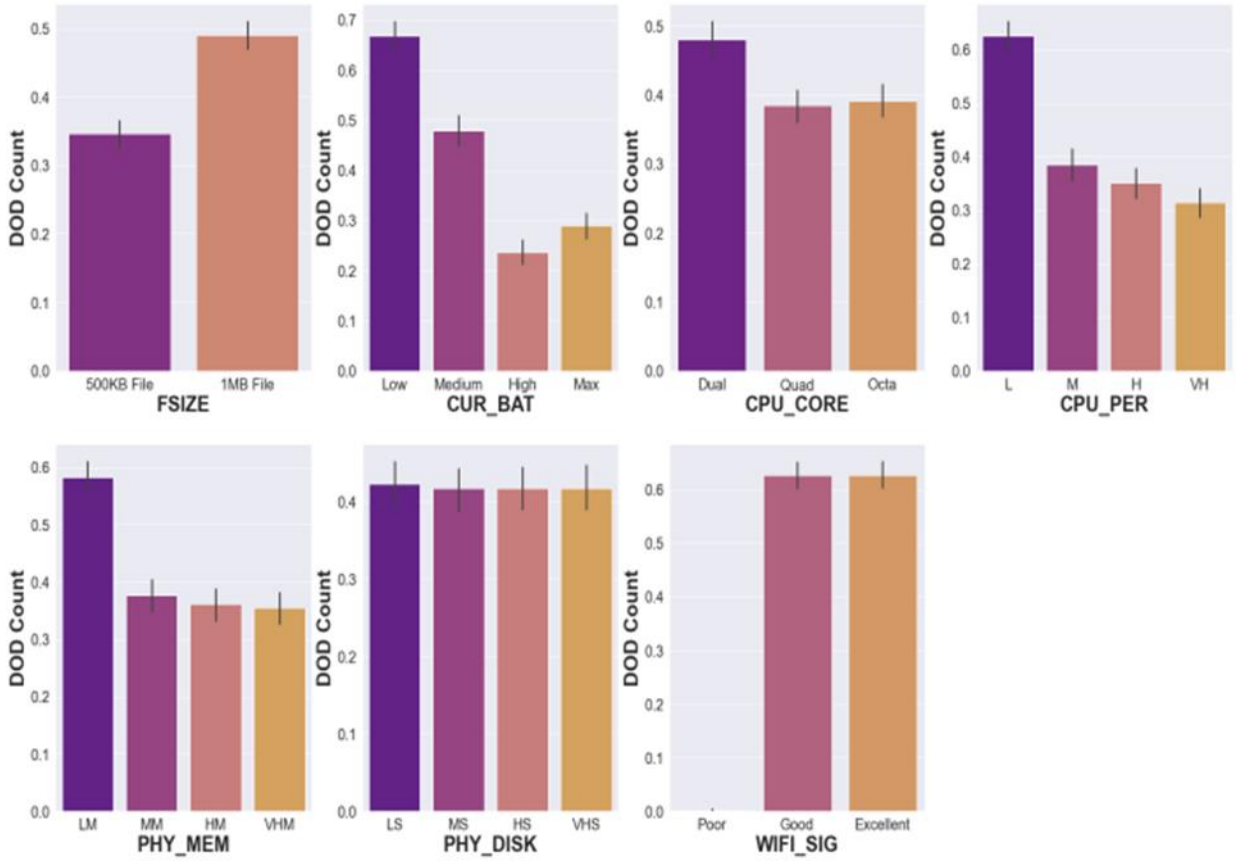

Figure 3 Feature-Target Correlation Visualization

Figure 3 represents the relationship between input feature selection and output target feature. The correlation shows the capability of selecting the attributes that can make the predicted target variable value high accurate or removing the irrelevant attributes which can decrease the classification accuracy and quality of the model. It is observed that all the features of the normalized data set have a positive impact on increasing decision accuracy.

4.1. Optimal Data Offload Decision Tree Classifier

Decision tree classification algorithms are supervised machine learning algorithm that empowers predictive models with high accuracy, stability and ease of interpretation. These algorithms work best when a predefined target variable is present and can be applied for both categorical and continuous input/output variables. They are adaptable to solve classification and regression problems. In the ICODA framework, we have a data set with a categorical target variable - Data offload decision (DOD).

4.2. ID3 – Decision tree classifier

The Iterative Dichotomiser 3 (ID3) classification model constructs a decision tree by placing an attribute with low entropy or high information gain at the root spot of the decision tree. The workflow of ID3 classifier is presented in algorithm 7. The entropy (E) is evaluated as shown in Eq. (3),

$$E = \left[-\left(\frac{Pds}{Pds+Nds}\right) \times \log2 \left(\frac{Pds}{Pds+Nds}\right)\right] + \left[-\left(\frac{Nds}{Pds+Nds}\right) \times \log2 \left(\frac{Nds}{Pds+Nds}\right)\right] \quad (3)$$

Where 'Pds' represents the count of positive samples, 'Nds' represents the count of negative samples and 'Pds + Nds' represents total instances in the given mobile client resource

RESEARCH ARTICLE

consumption status training data set. The Information gain (IG) is evaluated as shown in Eq. (4),

$$IG = \text{Entropy (parent)} - [\text{Average Entropy (children)}] \quad (4)$$

Algorithm: ID3 Decision Tree classifier

Input: Profiled client device resource status report

Output: Local execution/Remote execution/No Execution decision

BEGIN

Compute *entropy* for data of client device resource status report.

For each data attribute

Compute *entropy* for all its possible categorical values.

Compute *information gain* for the data attribute.

Take out the attribute with *high information gain*.

Repeat the cycle until the desired data offload decision tree is generated

END

Algorithm 7 ID3 Decision Tree Classifier

4.3. CART - Decision Tree Classifier

The Classification and Regression (CART) model constructs a decision tree by finding the best split through the calculation of the weighted sum of Gini Impurity (GI) for both child nodes as shown in Eq. (5). This is repeated for all possible splits and then takes the one with the lowest Gini Impurity as the best split. Lower the value higher the purity and homogeneity of the nodes in a tree. The workflow of CART Decision Tree classifier is presented in algorithm 8.

$$GI = 1 - \sum(\text{success probabilities for each class})\text{^}2 \quad (5)$$

Algorithm: CART Decision Tree classifier

Input: Profiled client device resource status report

Output: Local execution/Remote execution/No Execution decision

BEGIN

The select Root node(S) is based on Gini Index and Maximum Information Gain.

Calculate the Gini Index and Information gain.

Select the node based on Minimum Gini Index or Maximum Information Gain.

Split set S to produce the subsets of data.

Recur on each subset.

Create the decision Tree.

END

Algorithm 8 CART Decision Tree Classifier

4.3.1. CART with Grid search Cross-validation (CART-GSCV)

It represents the application of the Grid search cross-validation method which tunes hyper-parameters to provide a better estimation of the performance of the CART model for every combination of parameters per grid. The parameters of importance are criterion :("Gini", "entropy"), splitter, maximum depth, minimum samples for split, minimum samples at leaf level.

4.4. Random Forest (RF) – Decision Tree Classifier

The Random forest (RF) classifier is an ensemble learning technique used to construct multiple CART models with different data instances and unique initial variables. The RF classifier chooses the classification that has the most votes among all the trees in the forest. The two major benefits of ensemble models are more accurate predictions and the procedure of combining multiple CART models into a single strong model that helps to provide productive decision can stabilize the overall machine learning model.

4.4.1. Random Forest Classifier with Randomized Search Cross-Validation (RF-RSCV)

It represents the application of the Random search cross-validation method which tunes hyper-parameters to provide a better estimation of the performance of the RFC model by selecting a random grid combination of parameters. Random search can draw hyper-parameter values from continuous distributions, allowing it to sample the parameter space more fully and efficiently. The hyper-parameters include maximum depth, minimum sample for split, maximum leaf nodes, minimum leaf samples, n-estimators, and maximum features.

4.4.2. Random Forest Classifier with Grid Search CROSS-Validation (RFC-GSCV)

It represents the application of the Grid search cross-validation method which tunes hyper-parameters to provide a better estimation of the performance of the RF Classifier model for every combination of parameters per grid. The parameters of importance are maximum depth, minimum sample for split, maximum leaf nodes, minimum leaf samples, n-estimators, and maximum features.

4.5. Performance Evaluation Metrics

We have considered the Classification accuracy, Precision, Recall, and F1 score as the prime performance factors to evaluate the quality of the training model.

RESEARCH ARTICLE

4.5.1. Classification Accuracy (CA)

It is the ratio of the number of correct predictions to the total number of predictions made out of all input samples.

$$CA = \frac{\text{True Positives (TP)+ True Negatives (TN)}}{\text{TP+TN+False Positives+False Negatives}} \quad (6)$$

4.5.2. Precision

It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives+False Positives}} \quad (7)$$

4.5.3. Recall

It is the number of correct positive results divided by the number of all relevant samples.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}} \quad (8)$$

4.5.4. F1- Score

F1 Score is the harmonic mean of precision and Recall used to find the balance between them. Since it tends to mitigate the effect of large outliers and intensify the smaller ones, it is the most effective evaluation metric in the classification framework to make it precise and robust.

$$\text{F1 Measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}} \quad (9)$$

On training the different machine learning classification models such as ID3, CART, and Random forest algorithms with real-time normalized data set, we have the following comparison graph in terms of classification accuracy as shown in Figure 4.

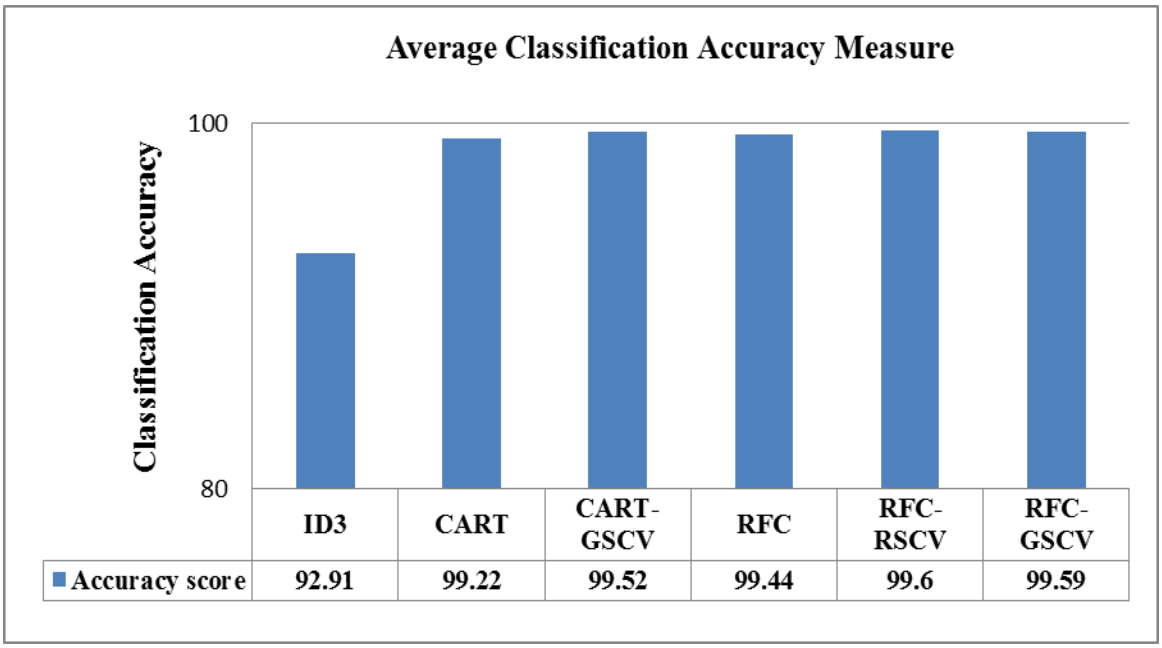

Figure 4 Comparison between ID3, CART, and RF Decision Tree Classification Accuracy

Form Figure 4, the random forest classifier can be considered as a highly accurate and robust method. And it does not suffer from the over fitting problem since it takes the average of all the predictions and remains to be unbiased. In the ICODA framework, it is noted that the classification accuracy of ID3, CART and RF models is at 92.91%, 99.22% and 99.44% respectively on applying Eq. (6). On cross-validation with Random search and Grid search, the RF classification accuracy estimation is at 99.60% and 99.59% respectively which implies the best quality training model performance for the data set on applying Eq. (6). Therefore the Random Forest Classifier is declared as optimal decision tree algorithm that outperforms all the other decision tree classifiers in the CODA framework. To support the result, we have also applied Eq. (7), Eq. (8), and Eq. (9), to measure Mean precision, Recall, and F1 score for the optimal random forest classifier which is at 0.9967 to indicate a significant classifier model performance. At this point, the successful WLAN connection between the smart mobile client and the centralized server is set. The medium-range data offload operation has been carried out with multiple compute-intensive task files that can keep the processor busy at all the available number of cores to measure execution time. The experiment has been repeated several times to get the productive result out of the real time device communication implementation. The graphical view of results obtained and their meaningful interpretation is presented in the next section.

RESEARCH ARTICLE

5. RESULTS AND DISCUSSION

The three processor bound source code files namely PI-22.py, PI-44.py, and PI-88.py of size 542KB, 598KB, and 635KB which follow the master-slave technique to keep dual, quad, and octal CPU cores busy with maximum workload possible respectively, it is noticed that if these three files are run solely on mobile client device itself, then it need to spend more energy as presented in Table 9.

Instead, if the files are offloaded to a centralized server for computation, it will reduce the load on the processor of mobile client device. From Fig. 5 it is observed that the battery life sustained measure for each processor bound source file is 0.55, 0.62, and 0.65 percent respectively. The data transfer time is < 0.022 seconds for the files moved between different smart mobile devices as depicted in Table 10.

MEAN REAL BATTERY LIFE SPENT ON CLIENT DEVICE			
File Name	**File Size**	**If executed on Local Device**	**When Offloaded to Remote device**
PI-22.py	524B	0.69%	0.14%
PI-44.py	598B	0.79%	0.17%
PI-88.py	635B	0.83%	0.18%

Table 9: Battery Power Consumption Measure With or Without Data Offloading

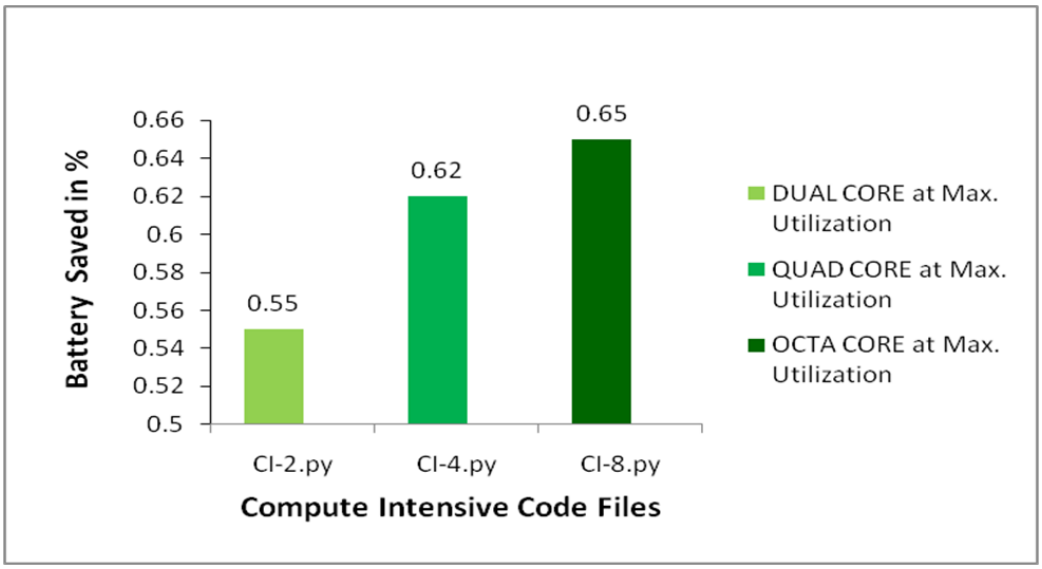

Figure 5 Battery Conservation in ICODA Offloading Scheme

File Name	File Size	Link Speed-in Mbps	Mean Data Transfer Time (secs)
PI-22.py	524B	96	0.017 s
PI-44.py	598B	96	0.019 s
PI-88.py	635B	96	0.021 s

Table 10 Data Transfer Time for Compute-Intensive Files

RESEARCH ARTICLE

MEAN REAL EXECUTION TIME MEASURE			
File_Name	**File_Size**	**Local CPU Time**	**On Offloading**
PI-22.py	524B	16.49 s	16.578 s
PI-44.py	598B	20.32 s	20.418 s
PI-88.py	635B	26.28 s	26.386 s

Table 11 Execution Time Comparison

The smart mobile device task execution time computation in ICODA framework on offloading considers several critical parameters such as CSLC search time, CSDOD time, Task transmission time from a mobile client device to centralized server (MC-CS), SSGC search time, Task transmission time from the centralized server to neighbor mobile device (CS-NMD), Neighbor mobile device (NMD) run time, Task result transmission time neighbor mobile device to a centralized server (NMD-CS) and Task transmission time from a centralized server to actual smart device client(CS-MC). From Table11, for task files, PI-22.py, PI-44.py, PI-88.py, the local device CPU time computed is 16.49s, 20.32, and 26.28s respectively. But on task data offloading in the ICODA framework, the task turnaround time is still evaluated to 16.578s, 20.418s, and 26.386s with the difference of about 0.088s, 0.098s, and 0.106s compared to local CPU Time is still a remarkable operation as shown in Table 12. The real advantage of the ICODA framework lies when we avoid data offloading using local or global data cache mechanisms which are as shown in Table 13.

CSLC Search Time	**CSDOD Time**	**TTT MC-CS**	**SSGC Search Time**	**TTT CS -NMD**	**NMD Run Time**	**TTT NMD -CS**	**TTT CS-MC**	**Task Turnaround Time**
0.012s	0.01s	0.016s	0.007s	0.015s	16.49s	0.016s	0.012s	16.578s
0.013s	0.01s	0.018s	0.008s	0.017s	20.32s	0.018s	0.014s	20.418s
0.014s	0.01s	0.02s	0.008s	0.019s	26.28s	0.02s	0.015s	26.386s

Table 12 Turnaround Time Computation for a Task

CODA framework	**Cache Hit?**	**Remarks**	**Task Turnaround time**
Client-side local cache (CSLC) search	PI-22 output file Found	Task execution time is equal to CSLC search time	0.012s
	PI-44 output file Found		0.013s
	PI-88 output file Found		0.014s
Server-side global cache (SSGC) search	PI-22 output file Found	Task execution time is a sum of (CSLC search time + CSDOD time + TTT MC-CS time + SSGC search time + TTT CS-MC time)	0.045s
	PI-44 output file Found		0.049s
	PI-88 output file Found		0.072s

Table 13: Turnaround Time Computation for a Task on Successful Cache Hit

RESEARCH ARTICLE

From Table 13, it is observed that there is a huge time difference in turnaround time measure since the task file is not physically executed but the corresponding output file is extracted from the cache to serve the client device execution request. Since the compute-intensive files consume less data communication time, it can lead to minimization of energy consumption on a client device and is preferable to perform a data offloading operation on these files for remote server execution. An artificial neural network back propagation algorithm is applied on both local and global cache data with sizes ranging from 1000 to 6000 entries and 3000 to 12000 entries respectively. The results of performance evaluation are evident to declare that the cache replacement prediction accuracy increases with the increase in training size as shown in Table 14 and 15. The Artificial neural network–back propagation model is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous iteration. This ensures the minimized error rate and enhanced generalization to produce a more reliable model. It uses all inputs for training and can make incremental updates with a stochastic gradient descent algorithm. It is useful for a continuous data set.

The Steps of ANN-back propagation Training Model:

1. Forward pass
2. Calculate error or loss
3. Backward pass

Algorithm: ANN-back propagation

Input: Cache data attributes

Output: Prediction accuracy

BEGIN

Assign random weights.

Find activation rate of hidden Nodes.

Find the activation rate of Output Nodes.

Find the error rate at Output Node.

Cascade and recalibrate the error in the backward process.

END

Algorithm 10 ANN-Back Propagation Model

ANN – Back Propagation Model Performance in CSLC				
Training set size	1000	2000	4000	6000
Learning Time (sec)	6.5	10.6	16.7	21.4
Prediction accuracy	88.7	92.2	94.1	96.4

Table 14 Local Cache Performance Evaluation

ANN- Back Propagation Model Performance in SSGC				
Training set size	3000	6000	9000	12000
Learning Time (sec)	11.5	18.1	23.7	27.9
Prediction accuracy	91.3	94.6	95.8	98.3

Table 15 Global Cache Performance Evaluation

Frame work	**Cloud Type**	**Client data Caching**	**System profiler**	**Turnaround Time on an Offload**	**Energy-Saving on Offload**	**Experiment**
ICODA	Private cloud / MEC	Yes	Run time	Lower	80 – 90%	Real-time
JAY (2021)	Custom	No	Run time	Higher	20 – 40%	Real-time

RESEARCH ARTICLE

OREGANO (2020)	MEC	No	No	Higher	71%	Real-time
ULOOF (2018)	MCC	No	No	Higher	36 %	Simulation

Table 16: Performance Evaluation of ICODA with JAY, OREGANO, and ULOOF Offloading Frameworks

Therefore, it is clear from Table 16 that the optimization in mobile client device work time is possible by adopting machine learning approaches in data offloads decision and data caching mechanism of ICODA framework and is promising to achieve better proficient performance with reduced run time.

6. CONCLUSION

In this paper, the proposed intelligent composite offload decision algorithm (ICODA) framework can outperform with optimized work time proficiency for two main reasons namely, a popular data caching mechanism to postpone the offloading operation as long as possible by keeping the most important results intact on storage and a random forest classifier algorithm with 99.44% best mean accuracy can optimize the length of the health of client mobile device work time. The data caching components of the proposed ICODA framework has been designed, implemented, and analysed. The performance of prediction accuracy using the ANN-back propagation machine learning model on local and global cache tables has been evaluated. The prediction accuracy was observed at 96.4% in the local cache and 98.3 in the global cache which is significant. Therefore, The ICODA framework is stable with infrastructure-less content caching at user equipment device level on a private network. A high bandwidth Wi-Fi-enabled network will be an added advantage for additional communication efficiency. This framework is generic and flexible in its architectural nature and can add any improvised decision-maker scheme to it for better results in the future for different applications and its real data sets. The extended experimental observation makes us realize that this framework can virtually double the energy of mobile devices since they are not used for workload execution unless the workload is new and unique in relevant to the specific organization private cloud giving rise to the substantial increase in overall mobile device work time enhancement.

REFERENCES

[1] Silva, Joaquim & Marques, Eduardo & Lopes, Luís & Silva, Fernando, "Energy-aware adaptive offloading of soft real-time jobs in mobile edge clouds", Journal of Cloud Computing, volume 10, Pages 1-21,2021.

[2] Wang, Yantong; Friderikos, Vasilis, "A Survey of Deep Learning for Data Caching in Edge Network" Informatics , volume 7, no. 4, Pages 1-29, 2020.

[3] Goncalo Carvalho, Bruno Cabral, Vasco Pereira, Jorge Bernardino, "Computation offloading in Edge Computing environments using Artificial Intelligence techniques", Engineering Applications of Artificial Intelligence, Volume 95, 2020.

[4] Sanches P., Silva J.A., Teofilo A., Paulino H, "Data-Centric Distributed Computing on Networks of Mobile Devices", Euro-Par 2020: Parallel Processing - Lecture Notes in Computer Science, volume 12247. Springer, Pages 296-311, 2020.

[5] T. Q. Dinh, Q. D. La, T. Q. S. Quek and H. Shin, "Learning for Computation Offloading in Mobile Edge Computing," in IEEE Transactions on Communications, vol. 66, no. 12, pp. 6353-6367, Dec. 2018.

[6] Z. Wen, K. Yang, X. Liu, S. Li and J. Zou, "Joint Offloading and Computing Design in Wireless Powered Mobile-Edge Computing Systems With Full-Duplex Relaying," in IEEE Access, vol. 6, pp. 72786-72795, 2018.

[7] Z. Chang, L. Lei, Z. Zhou, S. Mao and T. Ristaniemi, "Learn to Cache: Machine Learning for Network Edge Caching in the Big Data Era," in IEEE Wireless Communications, vol. 25, no. 3, pp. 28-35, June 2018.

[8] Farhan Azmat Ali, Pieter Simoens, Tim Verbelen, Piet Demeester, B Dhoedt, "Mobile device power models for energy-efficient dynamic offloading at runtime", Journal of Systems and Software, Volume 113, PP. 173-187, 2016.

[9] N. Fernando, S. W. Loke and W. Rahayu, "Computing with Nearby Mobile Devices: A Work Sharing Algorithm for Mobile Edge-Clouds," in IEEE Transactions on Cloud Computing, vol. 7, no. 2, pp. 329-343, 1 April-June 2019.

[10] S. Yu, R. Langar, X. Fu, L. Wang and Z. Han, "Computation Offloading With Data Caching Enhancement for Mobile Edge Computing," in IEEE Transactions on Vehicular Technology, vol. 67, no. 11, pp. 11098-11112, 2018.

[11] J. L. D. Neto, S. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar and S. Secci, "ULOOF: A User Level Online Offloading Framework for Mobile Edge Computing," in IEEE Transactions on Mobile Computing, vol. 17, no. 11, pp. 2660-2674, 2018.

[12] R. Aldmour, S. Yousef, M. Yaghi and G. Kapogiannis, "MECCA offloading cloud model over wireless interfaces for optimal power reduction and processing time," 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation pp. 1-8, 2017.

[13] S. Ahn, J. Lee, S. Park, S. H. S. Newaz and J. K. Choi, "Competitive Partial Computation Offloading for Maximizing Energy Efficiency in Mobile Cloud Computing," in IEEE Access, vol. 6, pp. 899-912, 2018.

[14] Yiming Miao, Gaoxiang Wu, Miao Li, Ahmed Ghoneim, Mabrook Al-Rakhami, M. Shamim Hossain, "Intelligent task prediction and computation offloading based on mobile-edge cloud computing", Future Generation Computer Systems, Volume 102, Pages 925-931, 2020.

[15] P. Nawrocki, B. Sniezynski, H. Slojewski, "Adaptable mobile cloud computing environment with code transfer based on machine learning", Pervasive and Mobile Computing, Volume 57, Pages 49-63, 2019.

[16] Hyun-Woo Kim, Jong Hyuk Park, Young-Sik Jeong, "Adaptive job allocation scheduler based on usage pattern for computing offloading of IoT", Future Generation Computer Systems, Volume 98, Pages 18-24, 2019.

[17] Abdulhameed Alelaiwi, "An efficient method of computation offloading in an edge cloud platform", Journal of Parallel and Distributed Computing, Volume 127, Pages 58-64, 2019.

[18] Lee, Hochul ; Lee, Jaehun ; Lee, Young Choon ; Kang, Sooyong, "CollaboRoid : Mobile platform support for collaborative applications", Pervasive and Mobile Computing, Vol. 55. pp. 13-31, 2019.

RESEARCH ARTICLE

Authors

Mr. Sridhar S K, Research scholar, working as an Assistant professor in Computer Science and Engineering department, BITM, Ballari. His research area is Machine learning assisted mobile cloud computing. He has undergone expert level workshop on embedded systems by Wipro in 2016, received IBM certification on Mobile application development in 2017, He has received Inspire Faculty Excellence award in 2018 and acquired knowledge on Research Methodology & LaTex by VTU in 2019.

Dr. Amutharaj Joyson is a Professor and Head in the Department of Information Science Engineering at RajaRajeswari College of Engineering, Bengaluru. He has received his Ph.D. doctorate from Anna University in 2012. He was granted with two International Patents by IP Australia, Australian Government. He also published four Indian Patents with Controllers of Patents, Chennai Office, and 21 research papers in refereed International Journals and presented 12 papers in International Conferences which includes IEEE and ACEEE conferences. Currently he is guiding three Ph.D. scholars in the areas of Mobile Cloud Computing, Sensor Networks and Content Distribution Networks. He has acted as technical programme committee facilitator and reviewer for International Journal of Network and Computer Applications (JNCA), Computer and Communications, Elsevier Publications, International Journal of Computer Science and Information Security (IJCSIS) - USA, IEEE sponsored Conferences and reputed Scopus indexed journals. He is a fellow member of the Institution of Engineers (IE), Institution of Electronics and Telecommunication Engineers (IETE), a life member of Indian Society for Technical Education (ISTE), Life member of International Association of Engineers (IAENG) and also the Life member of International Association of Computer Science and Information Technologists (IACSIT).

How to cite this article:

Sridhar S K, J. Amutharaj, "Optimization of Smart Mobile Device Work Time Using an Optimal Decision Tree Classifier and Data Caching Technique in on Premise Network", International Journal of Computer Networks and Applications (IJCNA), 8(6), PP: 702-718, 2021, DOI: 10.22247/ijcna/2021/210720.