3. QOS-AWARE LOAD BALANCING

3.1 INTRODUCTION

In MCC environment [154], the mobile application requires a high level of responsiveness, and it demands intensive computing resources in a mobile device to execute the sophisticated applications. The remote execution and code offloading in MCC environment have created a significant impact on the capabilities of smartphones and availability of cloud remote servers. The major impacts on MCC framework related to local execution, computation offloading, and remote execution are load balancing, task scheduling, and resource selection.

Load balancing is the process of balancing the load between schedulable tasks of mobile applications with selected cloud resources. Balancing the load between the tasks and resources are impacted from two broad main categories: task scheduling and resource allocation. Task scheduling in the mobile cloud is crucial for executing the requests of a similar application from multi-tenants. In resource allocation [155], the tasks are allocated to mobile core processors or available cloud resources. Task scheduling management increases resource consumption and reduces turnaround time during the task execution on a mobile device and the cloud.

QoS is the mutual attempt of service performance that decides the user’s degree of satisfaction with a particular service. To preserve the device energy and improve the user experience, the QoS-aware load balancing approach (QALBA) implements task scheduling and resource selection using Enriched-Look ahead HEFT (E-LHEFT) algorithm by exploiting MAUI architecture and Pareto principle. Furthermore, this chapter presents the implementation results of the QALBA methodology with the baseline methods.

3.2 AN OVERVIEW OF EXISTING LOAD BALANCING APPROACHES

TS-QoS [64] algorithm prioritizes the tasks depends on the special attributes of the tasks and achieves load balancing in cloud computing. It
performs dynamic prioritization to adopt FCFS strategy using expected completion time of the task. Green spot algorithm [100] preserves the energy-efficient battery level of the mobile device for various cloud-based mobile applications. Genetic algorithm (GA) [67] based scheduling strategy achieves load balancing and reduces dynamic migration. HACAS [106] and PA-LBIMM [70] ensures in satisfying the users’ demand with high QoS and load balancing. GA, TS-QoS, and HACAS algorithms initially perform scheduling followed by the load balancing but do not consider concurrent monitoring system. The conventional Look ahead variation of the Heterogeneous Earliest Finish Time (L-HEFT) algorithm [156] schedules the tasks of an application using precedence requirements of the tasks. Resource selection depends on only EFT (Earliest Finish Time) value of each task and corresponding children tasks in the task graph. Look ahead algorithm reduces the makespan about 20% compared to the HEFT algorithm due to the consideration of the currently selected task and also upcoming tasks. The L-HEFT algorithm does not work on the feasibility of a higher look ahead depth of the mobile applications. Even though the existing task scheduling algorithms provide QoS and achieve load balancing in a stringent mobile cloud environment, many priorities based scheduling techniques affect the low priority of users. The scheduling strategy suffers by the uncertainty when there is the waiting time of the task is high in the queue. Also, the existing migration based load balancing after scheduling the task increases the processing delay.

3.3 PROPOSED QOS AWARE LOAD BALANCING FRAMEWORK

The QALBA methodology implements the task scheduling, resource selection, and load balancing methods with high QoS in a mobile cloud environment using MAUI architecture. The research work describes QoS-aware load balancing for ChessOCR (Chess Optical Character Recognition) mobile gaming application on MAUI architecture. The QALBA approach considers the input task as the captured image of the chess board from books or magazines in ChessOCR mobile application. It aims at preserving
the energy of the mobile device while executing the mobile gaming applications using image processing on MAUI architecture.

The proposed approach partitions a requested application into several tasks for reducing the application completion time using MAUI. E-LHEFT algorithm prioritizes the task of an application using an upward rank value from the existing task of the task graph. In the cloud, the proposed E-LHEFT task scheduling algorithm selects the PM resource for prioritized cloud tasks. The minimum completion time for the current task and the consequent task forms a group using E-LHEFT scheduling algorithm for PM selection. Thus, the Pareto principle achieves load balancing of PM resources while mapping the task group on cloud resources. VM selection considers the minimum completion time of the task group of selected PM. Thus, the proposed algorithm ensures load balancing and reduces the overall application completion time while executing the tasks on both the mobile cores and cloud server. Thus, the proposed framework saves the smart phone’s energy during the execution of mobile applications.

Figure 3.1: E-LHEFT scheduling algorithm in MAUI architecture

Figure 3.1 shows the proposed E-LHEFT algorithm in MAUI architecture. The mobile cloud computing offloading method has different
architectures for executing the compute-intensive and resource-intensive applications. MAUI [54] is a mobile cloud application model that enables energy-aware offload of mobile code to the remote cloud infrastructure and maximize the benefits of code offload for today's smartphone devices. It provides a programming environment, in which, the methods of an application can be outsourced for remote execution using optimization framework. Also, MAUI continuously measures the network connectivity to the infrastructure, estimating its bandwidth and latency.

The QALBA approach executes the mobile user requested tasks in MAUI architecture. In MAUI architecture, smartphone comprises solver interface for providing interaction and taking an offloading decision, a profiler for analyzing the energy consumption and data transfer requirements, and client proxy for offloading the tasks. Likewise, the server has four components such as a profiler, server proxy, solver, and MAUI controller. In server side, the solver is the main decision engine of the MAUI that schedules the incoming requests using the E-LHEFT algorithm. The MAUI solver applies the data collected by the MAUI profiler component as input that determines which mobile methods should execute locally and which methods should execute remotely in the resource-rich cloud server. The main goal of the solver interface is to find out the application partitioning strategy that reduces the energy consumption of the smartphones. The MAUI controller handles allocation of resources for incoming requests and authentication to instantiate a partitioned application. Remote Procedure Call (RPC) is a protocol that requests a service from a program located on another system in a network irrespective of network details. MAUI architecture using E-LHEFT algorithm saves the mobile device energy and application completion time in a mobile cloud environment.

3.3.1 QALBA System model

The proposed work assumes smartphones as the mobile devices and ChessOCR as a mobile application. The mobile cloud environment allows the compute-intensive application execution outside the mobile devices, i.e., cloud. Mobile user offloads the input chess image to the cloud through the
network for moving next piece. The mobile device, receives the result of cloud task after the execution completes. The cloud has the higher speed, and computation ability compared with mobile devices. Mobile device decides to offload the tasks to the cloud based on the application load.

The mobile device consists of ‘n’ number of heterogeneous cores for processing mobile tasks of an application. The mobile device does not allow the parallel execution of tasks on heterogeneous cores. The cores in the mobile device can only execute or send one task at a time, and hence, preemption is not allowed on the mobile device. In contrast, the cloud server runs the number of compute-intensive tasks with preemption. The cloud can process a large number of tasks in parallel as the tasks are independent. Mobile device requires the outcome of previous task execution before starting the execution of the current mobile task of the core. MCC environment includes ‘m’ mobile devices, wireless channel, and cloud.

Each mobile application partition into a set of tasks \( T_i \) and \( T_i \left( E_{CT} \right) \) is the expected completion time of each task, where \( i = \{1,2,..., n\} \) in the cloud. Mobile device offloads only the compute-intensive tasks instead of all tasks of an application. Mobile device contains limited resources in terms of cores for executing the local (mobile) tasks of an application. The cloud server consists of \( j \) unlimited resources for scheduling the multi users’ tasks where, \( j = \{1,2,..., n\} \). \( T_{LENI} \) denotes processing requirements of each task in the mobile cloud. Each request of the application has a unique ID, file size, type of processing, and location. Task scheduling uses \( VM_j \) and \( PM_j \) resources in a mobile cloud environment. Each mobile application employs the Directed Acyclic Graph (DAG) that consists of ‘n’ number of tasks with entry and exit task. The finishing time of exit task represents the overall application completion time locally and remotely. Mobile cloud considers the entry task and exit task, such that entry task has no preceding task, and exit task has no subsequent tasks.

The proposed work exploits the DAG graph [150] for task prioritization and selection. VM and PM resources have a different processing speed of Million Instruction Per Second (MIPS) value on a cloud server. A load of each
PM depends on the total number of VMs and utilization of each VM resources. Mobile execution does not consume more energy during local task execution. Transmitting and receiving time of a task depends on the task size and rate of the channel.

3.3.2 ChessOCR mobile gaming application

ChessOCR application recognizes the input image of the chessboard in books or magazines using OCR. OCR recognizes the photo or scanned image into different required forms such as text recognition, face recognition, image processing, video streaming, and gaming applications. It uses the mobile autofocus camera for recognizing the chess diagrams from books or magazines. Each recognized chess image in Portable Game Notation (PGN) format. The most recently recognized chess image is saved as Forsyth-Edwards Notation (FEN) string. Figure 3.2 illustrates the working process of ChessOCR mobile application.

![Figure 3.2: Working process of Chess OCR application](image)

ChessOCR guides the next move in the chess game from the current position of recognized chess image. DroidFish is an Android application for the chess engine for retrieving the position and analyzing the chess images from chess database. After the chess image recognition, a green box will appear on the mobile screen for selecting the white/black to move next position from the captured chess piece’s position.
3.4 TASK SCHEDULING AND RESOURCE SELECTION FOR CHESSOCR APPLICATION IN MOBILE CLOUD ENVIRONMENT

At MCC, entities are mobile users, scheduling system and service providers in the cloud. This proposed approach introduces a scheduling strategy for dynamic scheduling of ChessOCR mobile application. It mainly focuses on task scheduling and resource selection with QoS and load balancing, to reduce makespan and latency and save mobile device energy using the proposed QALBA approach. Figure 3.3 illustrates the task scheduling and resource selection strategy for ChessOCR in a mobile cloud environment.

![Figure 3.3: Task scheduling and resource selection strategy in mobile cloud environment](image)

### 3.4.1 QoS-guaranteed load balancing

The HEFT scheduling algorithm has three phases such as task prioritization phase, task selection phase, and processor selection phase. Task prioritization phase relies on the descending order of upward rank value from DAG. Task selection phase selects the task from the prioritized list for
scheduling. The processor selection phase depends on the minimal execution time of the current task. In the Lookahead HEFT algorithm, processor selection phase differs from HEFT algorithm to further reduce makespan than HEFT algorithm. It focuses on the current task and upcoming tasks while scheduling the tasks. However, the L-HEFT algorithm does not preserve the load balancing of the processors. EAPA [103] approach jointly schedules the tasks on mobile device cores, communication links, and cloud. EAPA employs HEFT algorithm for generating the minimum latency on heterogeneous processors in which processor represents the cloud resources. The prioritization phase and task selection phase of the proposed QALBA and the existing EAPA are similar.

The proposed E-LHEFT algorithm enhances L-HEFT algorithm in computation offloading, task scheduling, and load balancing and modifies the processor selection phase in physical machine and virtual machine selection in cloud to preserve the load balancing and reduces makespan of the application.

This work suggests an enhancement in the process of look-ahead HEFT task scheduling, by considering the information about EFT of all successor requests and forms a task group based on the closer or similar EFT. Moreover, the proposed work uses the Pareto principle while mapping the task group with the PMs to achieve load balancing. The Pareto principle considers the length of the task group and a load of the PM before selecting the PM for processing the task group. Thereby, the proposed work performs task scheduling, resource selection, and achieves load balancing simultaneously. Furthermore, VM selection depends on the minimum execution time of the VM for each task group.

3.4.1.1 Computation Offloading

The proposed QALBA approach employs the MAUI architecture to offload the intensive tasks from the resource-poor mobile device to the remote server in an energy-efficient manner. In the mobile cloud environment, many users run the mobile applications with the support of
dynamic offloading enabled mobile cloud architecture to reduce the burden of the mobile device. The offloading strategy partitions, each application into a subset of tasks including mobile and cloud tasks to improve the performance and save the mobile device energy. Computation Offloading migrates resource-intensive computation of mobile applications from portable devices to the cloud server using threshold value. The threshold value is based on file size of the application, the number of similar application requests and number of mobile cores with different processing capabilities. Mobile task and cloud task separation rely on the threshold value of task length from tasks of an application. Task consider as the cloud task if the length of the task is higher than the threshold length \( T_{LENi} > \alpha \) that tasks must have higher execution time on mobile devices than cloud server; otherwise \( T_{LENi} < \alpha \), the task is considered as a mobile task. The cloud task has the lowest expected execution time \( (t^C_i) \) on the remote server than minimum execution time \( (t^M_i) \) of mobile device. Expected cloud execution time is the summation of time taken for sending, receiving, and executing the task.

### 3.4.1.2 Task prioritization and selection

Task scheduler schedules the applications to the available resources for execution of the mobile applications. Initially, the cloud task scheduling contains the following phases: task prioritization, task selection, and task grouping. In cloud task scheduling, compute-intensive task prioritization based on the E-LHEFT algorithm upward rank value of tasks in the DAG. The upward rank of each task depends on the weight of the task \( (W_i) \) and a maximum rank of the \( \text{i}^{\text{th}} \) task successor. The equation 3.1 shows the upward rank value calculation for each cloud task. The weight of the cloud task \( (W_i) \) is the average execution time of each cloud task \( (t^C_i) \) on heterogeneous processors.

\[
R_U(T_i) = W_i + \max_{T_{i+1}\in\text{succ}(T_i)} R_U(T_{i+1}) \tag{3.1}
\]

Task priority is the descending order rank value of the task graph of exit task that rank value is the weight of the exit task either on a mobile
device or a cloud. The E-LHEFT algorithm finds minimum completion time of task and successor tasks on heterogeneous processors. The proposed algorithm modifies resource selection based on the minimum completion time for the selected task group on cloud resource and selected task and upcoming tasks on mobile cores. Task scheduler selects the tasks based on the descending order rank value (lowest rank value has the highest priority).

Task scheduler calculates the overall length of each task group and a minimum expected completion time of each task group on each VM resource using MIPS value. Scheduler contains the information of all active and inactive servers in the cloud with the information about entering, processing, and leaving time for the tasks. Load balancer estimates the expected load of each server using resource consumption of selected task for executing the application. Time taken by a mobile device to send a task to the cloud and receive the response from cloud through a wireless channel is given as in equation 3.2 and equation 3.3 respectively.

\[ t_i^t = \frac{T_i^{\text{size}}}{R^t} \quad (3.2) \]

\[ t_i^r = \frac{T_i^{\text{size}}}{R^r} \quad (3.3) \]

The task size differs from the transmitting, and the receiving task of the mobile device and the channels transmitting, and receiving rate are also different. The start time of each task depends on the finish time of preceding tasks in mobile cores. On the remote server, heterogeneous processors execute a group of tasks at a time.

MAUI architecture improves the performance level regarding energy consumption and memory on mobile devices. The proposed load balancing approach achieves minimum makespan using the expected completion time for scheduling the cloud tasks using an E-LHEFT algorithm. Mobile execution of tasks runs on different cores based on the processing speed of the processor.
3.4.1.3 Task grouping

In cloud server, the compute-intensive tasks are combined as the task groups based on the earliest finish time of the current task and upcoming tasks during PM selection. Similar EFT of tasks is divided into higher and lower length task groups using threshold value. The E-LHEFT algorithm selects the PM based on the length of the task group. Task group depends on (two level of High and Low) the load of the prioritized task, i.e. task on each core of the mobile device and the task group on each cloud resource. PM selection also satisfies the Pareto principle with the intention of maintaining the load balancing. The overall length or load of low length task group must not exceed the overall load of the high task group. The proposed algorithm not only provides QoS, but also ensures the load balancing of the server in a mobile cloud environment.

3.4.1.4 Processor selection: Local execution

In resource selection, the schedulable task group and tasks are allocated into available cloud resources and mobile cores respectively. In the proposed system, task scheduler assigns the mobile tasks to appropriate mobile core that has a minimum completion time of the task. The core group acts as a single execution unit with numerous performance or power levels. An appropriate core is dynamically selected to run the user task based on the application behavior and user-defined policies. This is achieved by transparently moving the task’s execution to that appropriate core and allowing other inactive cores into an idle state to conserve power [157].

Mobile task execution depends on the precedence requirements of \( T_i \), the completion time of the previous tasks to start the execution of the current task in the mobile core. In local execution, mobile task selects high processing speed of the cores using minimum completion time of the task. Mobile device executes the selected task after receiving the output of the predecessor task \( T_{i+1} \) if the predecessor task is executed in the cloud. The mobile device starts the selected task execution after the finishing time of
predecessor task $T_{i+1}$ if the predecessor task is executed in the local mobile device.

Each mobile task waits until the completion of its previous task if various mobile tasks run on different cores in mobile devices. Less-intensive tasks execute on multi-cores of the mobile device with the knowledge of expected completion time of tasks to save the mobile device battery level. On local execution, starting time of each task depends on the finishing time of a preceding task of an application on heterogeneous mobile core processors. The mobile device does not consume more energy during local task execution.

3.4.1.5 Processor selection: Remote execution

The remote server executes the offloaded task on the cloud i.e. cloud task. In remote execution, the processor selection considers the higher and lower length task groups. The cloud task execution also depends on the precedence requirements of the tasks in the DAG. The cloud starts the execution of $T_i$ after receiving the output of mobile task $T_{i+1}$ if preceding task $T_{i+1}$ is executed on mobile devices. Cloud task selection depends on the expected completion time of tasks in prioritized tasks. The equation 3.4 shows the formula for calculating the weight of average earliest completion time ($ECT_w$) of each task on heterogeneous processors in a cloud environment. E-LHEFT algorithm based prioritization depends on traversing of DAG from the exit task of an application.

\[
ECT_w = \frac{\sum_{T_{i+1} \in T_L} (R_U(T_{i+1}) \times ECT_{i+1})}{\sum_{T_{i+1} \in T_L} R_U(T_{i+1})} \tag{3.4}
\]

Where,

- $R_U(T_{i+1})$ - Upward rank value of each successor cloud task
- $ECT_{i+1}$ - Earliest completion time of preceding task $T_{i+1}$
- $T_L$ - List of tasks of an application
The equation 3.5 shows the overall cloud task completion time \( t_i^C \) depends on the transmitting \( t_i^t \), receiving \( t_i^r \), and cloud execution time \( t_i^c \) of the tasks.

\[
t_i^C = t_i^t + t_i^c + t_i^r
\]  

(3.5)

3.4.1.5.1 Pareto principle

Pareto principle based task-group-mapping of cloud resources maintains the load balancing based on the length of the task group and server utilization or current load on the server. Each application has the sequence of tasks that executes on cloud and mobile device. For instance, each offloaded move to the next position in a chess game is the task to the cloud. Several mobile users offload the tasks at the same time in which similar EFT of tasks is considered as a group for execution.

In proposed QALBA approach, task grouping, and Pareto principle process simultaneously to select the PM resource for the task execution. The proposed approach divides the length of the task group as high and low. Similarly, utilization of PM is categorized into high and low utilization. Pareto principle depends on the 80/20 rule. The proposed algorithm states that 80% of the length of the task group schedules on 20% of PM utilization or load, and 20% of length of task group schedules on 80% of PM utilization or load. Hence, it maintains the load balance of all active servers while executing the application in a cloud environment.

3.4.1.5.2 Physical machine selection

In the proposed approach, the selection of physical machine resources simultaneously satisfies load balancing using the Pareto principle for cloud task execution. Initially, task scheduler collects all the information on task groups and cloud resources if the mobile user submits their tasks for scheduling. According to the Pareto principle, the task group selects the cloud server among all active servers based on the load of the task group.
and appropriate PM resource. The existing algorithms [59] [65] [66] [101] for load balancing does not consider the dynamic load deviation value during PM selection. Hence, the load imbalance is occurring between physical machine resources and degrades the overall performance of the system.

### 3.4.1.5.3 Virtual machine selection

The proposed algorithm selects the virtual machine resources from that selected PM using expected completion time of the task group and MIPS of each VM. VM selection relying on the minimum completion time for the cloud task group on each VM among all VMs in selected PM. The scheduler selects the cloud VM resources based on GT\_LEN of each cloud task group from the prioritized cloud task list. Task scheduler calculates the expected completion time of task group using equation 3.6 and completion of each task group depends on the total length of the task group, file size, and MIPS value of VM on mobile cloud.

\[
T_i(E_{CT}) = \frac{\text{Total length of task group}}{VMPe\times VM\_MIPS} + \frac{\text{Input file size}}{VM\_BW} \quad (3.6)
\]

Where,

- \( VMPe \) - VM processing element
- \( VM\_MIPS \) - VM Millions of Instructions Per Second
- \( VM\_BW \) - VM bandwidth

The task group has the setup of Pareto principle based two main levels, such as GT\_Lm (H) and GT\_Lm (L). Task scheduler allocates GT\_Lm(H) task group on the high processing speed of the VM and GT\_Lm (L) task group on high utilization or a load of the PM than other PM utilization. The proposed E-LHEFT task scheduling algorithm for MCC is shown in Algorithm 4.1 depends on QoS and load balancing. Task scheduler compares total processing requirements of the task group (Total\_GT\_LEN) with the processing capabilities (RT\_LEN) of the cloud VM resource for equating the task group and cloud resources. If the expected completion time of each task group of virtual machine resources satisfies the following condition on equation 3.7, the task
has a preference for scheduling in the corresponding VM otherwise the task is switched over to another VM that has high processing specifications.

\[ VM_j^i (T_i(E_{CT})) \leq VM_{JS} \tag{3.7} \]

\( VM_{JS} \) represents the specification of VM in which specification denotes the VM processing speed, availability, and capability (\( VM_{MIPS}, VM_{Pe}, VM_{BW} \)) for scheduling the cloud tasks onto appropriate VM resources.

**Algorithm 3.1: E-LHEFT task scheduling algorithm for MCC**

**Input:** List of input tasks

**Output:** Scheduled tasks with QoS and load balancing

**Step 1:** Initially assign task groups \( m=0 \)

**Step 2:** Declare tasks of an application \( i=0 \) to total number of tasks (\( T_L_{size} \))

**Step 3:** Declare cloud and mobile resources \( j=0 \) to total number of available cloud resources (\( R_L_{size} \))

**Step 4:** Compare processing requirements of each task (\( T_{LENi} \)) with threshold value (\( \alpha \))

If (\( T_{LENi} > \alpha \)) then tasks of an application as cloud tasks (\( T_i^C \)) or (\( T_{LENi} < \alpha \)) then tasks of an application as mobile tasks (\( T_i^M \))

// **E-LHEFT task scheduling algorithm**

**Step 5:** Rank tasks of an application using upward rank value of each cloud task \( R_U(T_i) \) from DAG

For unscheduled tasks, find out highest \( R_U(T_i) \) from \( \{R_U(T_i)\} \) and put into selection priority variable

// **Task grouping and Pareto principle based PM selection**

**Step 6:** Assign children of selected cloud task to task list of an application \( \{T_L\} \)

**Step 7:** Consider cloud tasks from \( \{T_L\} \)

**Step 8:** Check total lower length of task group with total higher length of task group \( \{Total_GT LEN (L) \leq Total_GT LEN (H)\} \)

Groups current task and similar execution time of children tasks as \( GT_{Lm} \)

Splits \( GT_{Lm} \) into lower length task group \( GT_{Lm} (L) \) and higher length
task group GT\textsubscript{Lm}(H)

**Step 9:** Consider all tasks to schedule and all resources

**Step 10:** Find expected load (utilization) of PM

Assign GT\textsubscript{Lm}(L) \rightarrow \min (PM\textsubscript{j}^{U}) \forall \text{active PMs}

Assign GT\textsubscript{Lm}(H) \rightarrow \max (PM\textsubscript{j}^{U}) \forall \text{active PMs}

// VM selection

**Step 11:** Calculate expected completion time of each task T\textsubscript{i}(ECT)

**Step 12:** Consider length of task group GT\textsubscript{LEN}, file size and VM specification VM\textsubscript{jS}

**Step 12:** Check (GT\textsubscript{LEN}=VM\textsubscript{jS} && VM\textsubscript{j}(T\textsubscript{i}(ECT)) = VM\textsubscript{jS}) then

Schedule tasks in task group list GT\textsubscript{Lm} into Virtual Machine resources VM\textsubscript{j}

3.4.1.6 Load balancing

The proposed approach employs MAUI architecture as it supports energy-constrained mobile cloud environment to an appreciable level. In a mobile cloud environment, load balancer schedules VM resources based on the hardware resource capabilities. In MCC environment, scheduling strategy in [66] executes the load balancing after task scheduling and resource allocation. This scheduling results in load imbalance on VMs. The proposed approach balances the load while scheduling the tasks if load variation is high in the dynamic mobile cloud environment. This is achieved by concurrent monitoring and analyzing the system using matching capabilities of the task group and resource capabilities of the Pareto principle.

Load balancer dynamically monitors the capability and availability of each server while task scheduler schedules the tasks. The proposed load balancer avoids the VM migration using current server load monitoring concept during execution of scheduled tasks among cloud resources. Initially, the system has no periodic information about VM resources and, therefore, it randomly selects free PM for mapping VMs. The proposed algorithm balances a load of VM resources based on historical information and the current state of the system if the number of VM involvement is high. The load
deviation of all resources considers a load of each node and number of utilized nodes i.e. resources. \( L_j \) and \( L \) denote a load of each node and an average load of all utilized resources. Hence, load balancing depends on the load deviation as given in equation 3.8.

\[
LB = \sqrt{\frac{\sum_{j=1}^{n} (L_j - L)^2}{n}} \tag{3.8}
\]

Where,

\( n \) – Number of virtual machine resources

The research work [66] migrates VM from one PM to another PM due to load imbalance of VM resources. It affects the migration cost due to the one-time scheduling of the tasks. In contrast, the proposed approach achieves load balancing by continuously monitoring the cloud resources for every second without migrating VM resources. The proposed algorithm evaluates the expected completion time \( (T_i (E_{CT})) \) of the task with available VMs capabilities for further minimizing the execution time. The speed of VM differs from each other, and the expected completion time depends on the task group size, VM capability, and CPU speed (GHz). The utilization status of each server is simultaneously observed using a monitoring system for the cloud environment.

Load balancer avoids the node imbalance of physical hosts using the Pareto principle based resource selection while scheduling. It verifies the available resources \( (CPU_A) \) and requested resources \( (CPU_R) \) before assigning the tasks for achieving the load balancing. Load balancer balances the resources until \( PM_1 (L) = PM_2 (L) \) with various tasks of an application execution. It selects the server resource with minimum utilization of active servers.

3.5 ENERGY CONSUMPTION

Mobile device energy consumption depends on the limited bandwidth, Round-Trip Time (RTT) to offload the mobile code to the cloud server. RTT is
the sending and receiving time of the task between the mobile device and the cloud. The input data size varies RTT of the application using Wi-Fi or 3G. MAUI architecture takes the data transfer time in tens of millisecond order for 10KB to 100KB data. The mobile device energy saving calculation based on the call graph in which $K_n$ is the variable for differentiating the execution location of cloud (remotely, i.e., $K_n = 1$) or mobile device (locally, i.e., $K_n = 0$). In call graph, each node represents a method and its computational and energy costs, and each edge represents the input file size of the method and energy consumed to transfer the output stage remotely. $t_i^M$ and $t_i^C$ denote the task execution time on the cloud server locally (M) and remotely (C). $m$ and $n$ are the methods in application execution using call graph. $B_{m,n}$ is the transfer time between ‘$m$’ method and ‘$n$’ method and $C_{m,n}$ denote its energy cost. The proposed approach satisfies the following equation 3.9 while using MAUI approach. Equation 3.9 calculates the amount of time spent to execute the mobile application.

$$\sum_{n \in N} \left( (1-K_n) \cdot t_i^M + K_n \cdot t_i^C \right) + \sum_{(m,n) \in E} \left( |K_m-K_n| \cdot B_{m,n} \right) \leq L_t \tag{3.9}$$

Energy consumption ($E_C$) is based on the energy cost and the application completion time including computation and communication time. In equation 3.9, the first term represents the computation time of all tasks in the application and the second term refers the communication time of all tasks of an application. Equation 3.10 computes the amount of energy consumed by the mobile device during application execution. To compute the energy consumption, the system requires the energy cost per time in J/s. ‘$C$’ and ‘$C_{m,n}$’ represents the cost of computation and communication respectively, which are based on the computational and communication resources.

$$E_C = C \cdot \left[ \sum_{n \in N} \left( (1-K_n) \cdot t_i^M + K_n \cdot t_i^C \right) \right] + C_{m,n} \cdot \left[ \sum_{(m,n) \in E} \left( |K_m-K_n| \cdot B_{m,n} \right) \right] \tag{3.10}$$

The proposed approach saves the mobile devices’ energy in MAUI architecture. Hence, it improves the performance of the mobile applications.
The latency reduction of mobile application execution improves the number of application processing per second in the cloud. In MAUI, the overall execution latency \( L_t \) varies within 5% of the local execution latency.

### 3.6 EXPERIMENTAL EVALUATION

To demonstrate the importance of the proposed approach, numbers of experiments were carried out in a Cloud simulator environment. The proposed approach compares with existing algorithms such as GA [67], TS-QoS [64], Min-Min, and PA-LBIMM [70] in a cloud environment. Furthermore, the proposed algorithm is compared with existing Green spot [100], EAPA [103] and HACAS [106] in a mobile cloud environment.

#### 3.6.1 Experimental setup

MCC environment is highly dynamic in nature with a surplus amount of mobile user requesting the similar application at a time. In this proposed work, CloudSim tool simulates the task scheduling based grouping method and load balancing model of cloud resources. The simulation environment simulates mobile cloud framework by mobile user, mobile application model, and remote cloud server to run large scale mobile applications. Cloud executes the resource-intensive and compute-intensive tasks of an application on the remote server, and mobile executes the part of the task of an application based on mobile resource availability. The location of task processing varies according to the current task processing in mobile and cloud. The Cloud Information Service (CIS) provides the information about available cloud resources and resource name, ID, the total number of machines, total processing elements, and processing capability of each machine. Table 3.1 shows the simulation parameters and simulation values were used in experimental evaluation. Experimental setup considers 30 ChessOCR applications with various specifications and each application partitioned into the number of tasks with different length. Makespan of application ranges from 0 second to 1 second. The processing capabilities of PM resources vary between 1800, 2000, 4000, 6000 MIPS.
Table 3.1: Simulation setup

<table>
<thead>
<tr>
<th>Simulation parameters</th>
<th>Simulation value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tasks</td>
<td>10-150</td>
</tr>
<tr>
<td>Input file size</td>
<td>10-1000MB</td>
</tr>
<tr>
<td>Average task length</td>
<td>250-12500KB</td>
</tr>
<tr>
<td>Number of requests</td>
<td>500-2500</td>
</tr>
<tr>
<td>Number of mobile cores</td>
<td>3-5</td>
</tr>
<tr>
<td>Number of physical machines</td>
<td>10-50</td>
</tr>
<tr>
<td>Number of virtual machine instances</td>
<td>100-1000</td>
</tr>
<tr>
<td>Number of mobile applications</td>
<td>30</td>
</tr>
<tr>
<td>Execution time</td>
<td>0-35 minutes</td>
</tr>
<tr>
<td>Percentage of load change</td>
<td>10-50%</td>
</tr>
<tr>
<td>CPU Speed</td>
<td>30GHz-300GHz</td>
</tr>
<tr>
<td>PM MIPS</td>
<td>1500-6000MIPS</td>
</tr>
<tr>
<td>VM MIPS</td>
<td>100-1500MIPS</td>
</tr>
<tr>
<td>VM memory</td>
<td>4GB-32GB</td>
</tr>
<tr>
<td>VM bandwidth</td>
<td>200-1200Kbps</td>
</tr>
<tr>
<td>VM processing element</td>
<td>2</td>
</tr>
</tbody>
</table>

3.6.1.1 Evaluation metrics

**Makespan:** It is the total time taken to complete a mobile application execution in the mobile, and the cloud server, representing the time difference between the finish time of the exit task and start time of an entry task.

**Energy level:** It refers that the battery (energy) level of the mobile device while running a specific mobile application in the mobile cloud environment. Hence, the energy consumption of the mobile device is based on application completion time and energy cost.

**Load balancing:** It denotes the percentage of load balancing process required by the system that is the percentage of load deviation among the allocated cloud resources.
**Average Resource utilization:** It is the average value of utilizing each cloud resource among the allocated resources while executing a specific application.

3.6.2 Experimental results and analysis

The experimental results demonstrate the performance variation of the QALBA approach while experimenting with the conventional task scheduling, resource allocation, and load balancing techniques. The performance variation can be revealed using four unique metrics such as Number of tasks Vs Makespan, Time Vs Energy level, Number of tasks Vs Load balancing, and Number of tasks Vs Average resource utilization.

3.6.2.1 Number of tasks Vs Makespan

![Number of tasks Vs Makespan](image)

Figure 3.4 shows the makespan value while varying the number of tasks and the corresponding numeric values are shown in the Table 3.2. Comparison with the existing algorithms proves that the proposed QALBA approach minimizes the makespan value significantly using grouping method and Pareto principle. The proposed E-LHEFT algorithm groups the task request for execution. Each task group contains a different number of tasks.
based on the task file size. Pareto principle based group of task processing enhances the performance of mobile devices in terms of energy and application completion time. Therefore, the proposed E-LHEFT algorithm minimizes the makespan value by 5.9% when increasing the number of tasks from 10 to 50 but, in the same case, the existing scheduling algorithms increases the makespan value by 6.4% to complete the same mobile application execution in the mobile cloud environment.

Table 3.2: Number of tasks Vs Makespan

<table>
<thead>
<tr>
<th>Number of tasks</th>
<th>Makespan (ms)</th>
<th>E-LHEFT</th>
<th>Min-Min</th>
<th>TS-QoS</th>
<th>HACAS</th>
<th>EAPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>42</td>
<td>75</td>
<td>68</td>
<td>65</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>68</td>
<td>92</td>
<td>86</td>
<td>79</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>98</td>
<td>125</td>
<td>120</td>
<td>115</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>124</td>
<td>183</td>
<td>162</td>
<td>152</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>165</td>
<td>210</td>
<td>193</td>
<td>182</td>
<td>175</td>
<td></td>
</tr>
</tbody>
</table>

3.6.2.2 Time Vs Energy level

Figure 3.5: Time Vs Energy level
Figure 3.5 shows the reduction of mobile device battery level based on time variants while executing the tasks in mobile cores and remote cloud server. Table 3.3 represents the implementation results in numeric values of Figure 3.5. The proposed E-LHEFT algorithm minimizes the energy consumption of the mobile device while executing the offloaded tasks in the cloud by grouping the tasks. Initially, MAUI architecture saves the mobile device battery level while scheduling the tasks. This scheme focuses on the Pareto principle based task grouping which shortens the overall processing time of an application. Therefore, the proposed E-LHEFT algorithm consumes only 14.4% of battery energy from the initial energy level when executing the mobile application with the duration of 35 minutes but, the EAPA and green spot consumes 19.82% and 18.03% respectively.

Table 3.3: Time Vs Energy level

<table>
<thead>
<tr>
<th>Time (Min)</th>
<th>Energy level(mWh)</th>
<th>E-LHEFT</th>
<th>Greenspot</th>
<th>EAPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5900</td>
<td>5805</td>
<td>5700</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5850</td>
<td>5780</td>
<td>5720</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>5850</td>
<td>5700</td>
<td>5650</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>5550</td>
<td>5400</td>
<td>5250</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>5480</td>
<td>5380</td>
<td>5200</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>5180</td>
<td>4850</td>
<td>4730</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>5050</td>
<td>4700</td>
<td>4570</td>
<td></td>
</tr>
</tbody>
</table>
3.6.2.3 Number of tasks Vs Load balancing

The number of mobile users and mobile user requests are not stable in the dynamic mobile cloud environment; hence, the load of each PM varies according to the number of mobile users' requests. The load deviation values of all resources are shown in Figure 3.6 and corresponding numeric values are shown in Table 3.4. From the experimental graph, it is observed that the load deviation of E-LHEFT algorithm decreases compared to existing algorithms. The E-LHEFT algorithm balances the load of PM resources using concurrent processes of the task scheduling and load balancing in a mobile cloud environment. Each physical host handles the load balancing based on the consideration of availability and utilization of the host. Therefore, Pareto principle based task-group-mapping of cloud resources minimizes the load deviation value in the proposed approach. In resulting, the proposed E-LHEFT algorithm maintains the load deviation within 75%, but the existing methods have the load deviation values between 77% to 92%, when the number of tasks is 50.
Table 3.4: Number of tasks Vs Load balancing

<table>
<thead>
<tr>
<th>Number of tasks</th>
<th>Load balancing (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E-LHEFT</td>
</tr>
<tr>
<td>10</td>
<td>49</td>
</tr>
<tr>
<td>20</td>
<td>54</td>
</tr>
<tr>
<td>30</td>
<td>61</td>
</tr>
<tr>
<td>40</td>
<td>68</td>
</tr>
<tr>
<td>50</td>
<td>75</td>
</tr>
</tbody>
</table>

3.6.2.4 Number of tasks Vs Average Resource Utilization

Figure 3.7 shows the average resource utilization that varies with the number of tasks for scheduling on cloud and mobile resources. Table 3.5 illustrates the numeric values of Figure 3.7. The proposed E-LHEFT algorithm increases the average resource utilization based on the task grouping and resource capabilities matching method using Pareto principle.
The proposed resource utilization depends on the task group size and started VMs on a particular server. The technique processes more number of applications by satisfying load balancing. Also, the E-LHEFT algorithm optimally utilizes the allocated resources rather than initiating a new PM and VM resources, which intends to reduce the energy consumption in cloud server. Accordingly, the E-LHEFT algorithm upholds the average resource utilization from 82% to 98%, but the existing algorithms maintain the average resource utilization from 64% to 88% alone when increasing the number of tasks in the mobile cloud environment.

Table 3.5: Number of tasks Vs Average Resource utilization

<table>
<thead>
<tr>
<th>Number of tasks</th>
<th>Average Resource Utilization (%)</th>
<th>E-LHEFT</th>
<th>TS-QoS</th>
<th>EAPA</th>
<th>HACAS</th>
<th>PA-LBIMM</th>
<th>Min-Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td>82</td>
<td>68</td>
<td>72</td>
<td>73</td>
<td>76</td>
<td>64</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>85</td>
<td>74</td>
<td>75</td>
<td>75.5</td>
<td>83</td>
<td>70</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>90</td>
<td>82</td>
<td>81</td>
<td>83</td>
<td>87</td>
<td>78</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>92</td>
<td>85</td>
<td>85</td>
<td>87</td>
<td>89</td>
<td>81</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>98</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>92</td>
<td>84</td>
</tr>
</tbody>
</table>

3.7 SUMMARY

- In this chapter, the design and development of MCC framework by enriching load balancing approach with task scheduling, and resource selection modules are presented.
- The proposed Enriched-Look ahead Heterogeneous Earliest Finish Time (E-LHEFT) algorithm for MCC framework using task prioritization, task selection, task grouping, Pareto principle, and processor selection are explained clearly.
- The Mobile Assistance Using Infrastructure (MAUI) mobile application model is utilized in QALBA approach for computation and data storage execution in outside of the mobile device.
• The significant performance improvement is attained by focusing on the QoS-aware mobile application execution on a remote server in mobile cloud environment.

• Eventually, it illustrates the evaluation results of the QALBA along with conventional mobile cloud task scheduling, resource allocation, and load balancing approaches when experimenting using the mobile gaming applications.

• In the proposed approach, the mobile device energy consumption is calculated using the energy profile model in MCC framework.

• The main contributions of this research work are computation offloading for multiple mobile user requests, task prioritization using an upward rank value for both current task and successor tasks, higher and lower length task groups, load balancing using Pareto principle, and processor selection in local (mobile cores) and remote execution (cloud server) in QoS-aware load balancing approach.

• The QALBA approach increases the provisioning of SaaS and PaaS services in the dynamic mobile cloud environment.

• The proposed E-LHEFT algorithm satisfies the QoS requirements in terms of minimizing the overall application completion time by 5.7%, significantly maintains the energy level of the mobile device by 14% to 19% on average, improves scalability and maintains the load deviation among physical and virtual machine resources within 75%, and maximizes the average resource utilization of cloud resources by 90% by processing more number of mobile client application requests.

Chapter 4 discusses the service level agreement based optimization approach to satisfy both the mobile users and cloud service providers in the proposed MCC framework.