IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 5, 2017, accepted January 24, 2017, date of publication February 14, 2017, date of current version March 13, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2669080

Heterogeneity-Aware Task Allocation in Mobile Ad Hoc Cloud

IBRAR YAQOOB[1], EJAZ AHMED[1], (Member, IEEE), ABDULLAH GANI[1], (Senior Member, IEEE), SALIMAH MOKHTAR[2], AND MUHAMMAD IMRAN[3], (Member, IEEE)

[1]Centre for Mobile Cloud Computing Research, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia
[2]Department of Information System, University of Malaya, Kuala Lumpur 50603, Malaysia
[3]College of Computer and Information Sciences, King Saud University, Almuzahmiyah, 11451, Saudi Arabia

Corresponding authors: I. Yaqoob (ibraryaqoob@siswa.um.edu.my) and A. Gani (abdullah@um.edu.my)

This work was supported by the Bright Sparks Program and the Research Grant from the University of Malaya under Grant BSP/APP/1689/2013, Grant RP012C-13AFR, and Grant UM.C/625/1/HIR/MOE/FCSIT/03. The work of Imran was supported by the Deanship of Scientific Research at King Saud University through Research under Grant RG 1435-051.

ABSTRACT Mobile *Ad Hoc* Cloud (MAC) enables the use of a multitude of proximate resource-rich mobile devices to provide computational services in the vicinity. However, inattention to mobile device resources and operational heterogeneity-measuring parameters, such as CPU speed, number of cores, and workload, when allocating task in MAC, causes inefficient resource utilization that prolongs task execution time and consumes large amounts of energy. Task execution is remarkably degraded, because the longer execution time and high energy consumption impede the optimum use of MAC. This paper aims to minimize execution time and energy consumption by proposing heterogeneity-aware task allocation solutions for MAC-based compute-intensive tasks. Results of the proposed solutions reveal that incorporation of the heterogeneity-measuring parameters guarantees a shorter execution time and reduces the energy consumption of the compute-intensive tasks in MAC. A system model is developed to validate the proposed solutions' empirical results. In comparison with random-based task allocation, the proposed five solutions based on CPU speed, number of core, workload, CPU speed and workload, and CPU speed, core, and workload reduce execution time up to 56.72%, 53.12%, 56.97%, 61.23%, and 71.55%, respectively. In addition, these heterogeneity-aware task allocation solutions save energy up to 69.78%, 69.06%, 68.25%, 67.26%, and 57.33%, respectively. For this reason, the proposed solutions significantly improve tasks' execution performance, which can increase the optimum use of MAC.

INDEX TERMS Mobile ad hoc cloud, mobile cloud, task allocation, mobile cloud computing.

I. INTRODUCTION

Recent advances in Mobile Cloud Computing (MCC) are motivating mobile users to leverage on the benefits of a plethora of novel applications, such as m-gaming, m-learning, and m-health [1]. To meet the requirements of various applications, MCC provides three different types of platforms to augment the resources of power-constrained mobile devices. These platforms are remote cloud, server-based cloudlet, and Mobile Ad Hoc Cloud (MAC) [2], [3]. Due to low computation time and on-demand availability of resources, mobile devices offload compute-intensive tasks to the remote cloud through wireless technologies [4], [5]. However, certain applications may suffer from high latency, jitter, and packet losses due to weak connectivity. To cope with these issues, server-based cloudlet relies on locally available high-end devices (i.e., servers) that may not be guaranteed always. The design philosophy of MAC is to form a group of nearby mobile devices and capitalize on their unexploited resources in order to execute compute-intensive tasks in case of weak or non-availability of connectivity or servers [6]. The mobile devices have to perform authentication, management, resource monitoring, and task allocation along with application execution in a distributed manner that involves processor cycles and energy [7]. The performance of the task execution primarily depends on the availability of resources which may vary from one participating mobile device to another i.e., heterogeneous MAC [8], [9].

Considering MAC heterogeneity, execution time and energy consumption can be minimized by incorporating the resource availability information (i.e., number of cores and applications running in the background) while allocating tasks to mobile devices. The faster the processor, the smaller

the execution time and, hence, lower energy consumption. Moreover, executing a large task on a mobile device with limited capability (e.g., processing) may lead to increased execution time and energy consumption. Furthermore, a mobile device running a large number of processes in background increases the overall application execution time.

This study aims to minimize the execution time and energy consumption by proposing five heterogeneity-aware task allocation solutions. The contributions of this paper are summarized as follows:

(a) First, we conducted an empirical study to investigate the impact of resource heterogeneity parameters on the performance of task execution and identify important device heterogeneity parameters, such as CPU speed, workload, and number of cores;
(b) We propose five heterogeneity-aware task allocation algorithms and a corresponding system model is developed;
(c) We validate the developed system model with the simulation results obtained from five heterogeneity-aware task allocation solutions;
(d) We perform evaluation by comparing the execution time and energy consumption results of five proposed heterogeneity-aware task allocation solutions with random-based task allocation solution.

The rest of the paper is organized as follows. Section II investigates work related to MAC. Section III presents problem analysis by discussing the performance evaluation methodology, the application that is used for analysis on heterogeneous environment, performance metrics, experimental parameters, and results and analysis. Section IV describes a system model considered for this work and also presents five heterogeneity-aware task allocation algorithms for MAC. Section V describes the experiment setup, performance metrics, and analysis of results. We validate the developed system model with the simulation results in Section VI. Section VII presents a mean value-based comparison of the proposed solutions with random-based task allocation in terms of execution time. Section VIII presents a mean value-based comparison of the proposed solutions with random-based task allocation in terms of energy consumption. Section IX discusses the scope and limitations of the work. Finally, we provide concluding remarks and future directions of the work in section X.

II. RELATED WORK

MAC is still in its infancy and as such very limited literature is available on the subject. We have already reviewed and presented most of the existing work related to MAC in [3]. For example, Li *et al.* [10] have addressed the problem of offloading compute-intensive applications, proposing a set of online and batch scheduling heuristics to offload independent tasks among participating mobile devices in a dynamic manner. For instance, MinHop heuristic takes into account the minimum number of hops from the client while allocating a task. Another heuristic (i.e., METComm) prefers to pick a mobile device that will take the minimum amount of time to execute the task. Similarly, minimum expected completion time is the preference of MCTComm during task assignment. Few other heuristics were proposed while keeping in mind the communication costs. Several performance metrics, such as average makespan, waiting time, slowdown, and utilization were used to validate the performance. The results advocated that the expected completion time must be taken into consideration during task allocation. Our work is more focused on device heterogeneity instead of communication cost.

A generic offloading framework for heterogeneous mobile cloud including cloudlet and MAC was proposed in [11]. It employed mobile devices, nearby cloudlets, and public clouds to improve the performance and availability of MCC services. However, the proposed framework is more generic and does not specifically consider the peculiarities of MAC during task allocation. To tackle the problem of MCC task allocation in heterogeneous wireless networks, Lu *et al.* [12] proposed offline centralized and online distributed schemes. First, they proved the problem as NP-hard. The aim of this work was to minimize average response time for an entire set of tasks. The proposed schemes considered various delays, such as communication, processing, and queuing during task allocation. Despite promising results, load balancing remains an issue.

To meet time constraints in local mobile clouds, an energy efficient task scheduling scheme was proposed in [13]. To schedule different tasks, the authors have proposed an adaptive probabilistic scheduler that not only satisfies time constraints but also minimizes energy consumption while executing compute-intensive real-time applications. The merits of the proposed scheduler include energy efficiency, scalability, and flexibility. However, complexity is one of the prime limitations.

An opportunistic ad hoc cloudlet service (OCS) mode was proposed in [14]. To offload compute-intensive tasks in a cost-effective and flexible manner, OCS adopts an intelligent and energy-efficient mechanism using ad hoc cloudlet.

To execute compute-intensive tasks, a distributed platform was proposed in [15] to employ resources of nearby mobile devices. To reduce energy consumption and computational cost, a task assignment mechanism was proposed in [16]. In addition, a two-stage Stackelberg game is also formulated to determine the execution units count that slave nodes are offering, while master node works on the compensation strategies based on the contributing resources. Nonetheless, resource and operational heterogeneity of mobile devices are not considered during task allocation.

Unlike most existing works, we consider resource heterogeneity during task allocation to improve performance in terms of execution time and energy consumption.

III. EMPIRICAL STUDY: IMPACT OF HETEROGENEITY ON TASKS' EXECUTION PERFORMANCE

To investigate the impact on the execution performance of compute-intensive tasks, this section presents

experimental setup, performance metrics, and crucial resource heterogeneity parameters.

A. EXPERIMENTAL SETUP

To analyze the performance, four mobile devices of different specifications are used. Compute-intensive tasks are given to run them on the mobile devices that helps to investigate the impact of heterogeneity on task execution time and energy consumption. Multi-threaded matrix multiplication and infinite loop execution applications are also developed. These mobile applications represent the class of compute-intensive applications. The following subsections further provides details of the experimental setup.

1) MOBILE DEVICE

A Samsung S II i9100g smart phone is used to conduct the experiment; specification details are provided in Table 1. We investigate the effect of heterogeneity on tasks' execution time and energy consumption by changing task sizes, CPU speed, workload, and number of cores of the mobile devices. To customize the CPU speed (e.g., 600MHZ, 800MHZ, 1008MHz, and 1200MHz) and number of cores (e.g., 1 and 2), two applications, Master CPU and Kernal Tuner, are respectively used. In addition, the Power Tutor application is used to measure the energy consumption.

TABLE 1. Specification of the Samsung S II i9100g.

Mobile components	Specification
CPU	Dual-core 1.2GHz
RAM	1 GB
OS	Android Jellybeans 4.1
Processor Architecture	ARMv7 rev3(v71)

2) MULTI-THREADED MATRIX MULTIPLICATION

A multi-threaded matrix multiplication application is designed to study the impact of heterogeneity of mobile device resources and workload on execution time and energy consumption when allocating tasks in MAC. The reason to design this application as a task instead of using a real-time task is time constraints. In the past, matrix multiplication has been used in image processing and MCC to perform an analysis of specified problems [17]. The application takes a set of the matrix as an input and gives a result after the multiplication. The application divides the matrix multiplication task and distributes it among the number of available mobile devices. The matrix multiplication is performed on the local device to compute the results. The applications and corresponding task sizes selected for the experiment are presented in Table 2. In addition, we develop an infinite loop application to analyze the workload impact on task execution time and energy consump-tion.

B. PERFORMANCE METRICS

The following performance metrics were used to analyze the impact of resource heterogeneity in MAC while allocating the task:

1) EXECUTION TIME

The execution time is defined as the number of seconds required to complete a task by the mobile device. This includes the time spent executing run-time or system services on its behalf. The execution time primarily depends on the task size, processor speed, and number of background processes running on a mobile device. Executing a compute-intensive task on a slow device may prolong execution time. Therefore, task allocation based on the mobile device specification can significantly improve performance.

2) ENERGY CONSUMPTION

Energy consumption is the number of millijoules (mJ) used to execute a task. Likewise, energy dissipation is proportional to the task size.

C. EXPERIMENTAL PARAMETERS

In the experiments, the following four crucial parameters (i.e., task size, workload, CPU speed, and number of cores) were used to investigate the impact of varying resource heterogeneity on the performance of task execution.

1) WORKLOAD

The amount of workload directly affects the execution time and energy. It is defined in terms of number of applications already running in the background on the mobile device. A mobile device with more workload will require more execution time and energy. Therefore, it is important to pick a mobile device with the least workload when allocating a task. We analyze workload by running the infinite loop application on a mobile device.

2) NUMBER OF CORES

The number of available cores in a mobile device significantly affects the execution time and energy as it supports parallelism. The availability of a greater number of cores can significantly reduce execution time. Hence, an effective and efficient task allocation strategy should consider the number of available cores in a mobile device.

3) PROCESSOR SPEED

The speed of the processor mainly determines the execution time and energy. The higher the speed of the processor, the quicker the execution time. Therefore, processor speed is one of the paramount concerns during task allocation, as mobile devices usually have different processor speeds.

4) TASK SIZE

The task size significantly impacts the execution time and the energy. As the task size grows, the execution time becomes longer, and hence, more energy is consumed. To ensure energy-efficient in-time execution, the size of the task is

TABLE 2. Details of the experimental parameters.

Parameters	Running Tasks	Mobile Settings using "CPU Master" Application
• Task Size	• 100 times 100×100 size matrices multiplication • 200 times 200×200 size matrices multiplication • 300 times 300×300 size matrices multiplication • 400 times 400×400 size matrices multiplication • 500 times 500×500 size matrices multiplication	• Scaling interactive 300MHz to 1200 MHz
• Workload (Running two, four, six, eight and ten applications)	• 300 times 300×300 size matrices multiplication	• Scaling interactive 300MHz to 1200MHz
• Processor speed (300MHz, 600MHz, 800MHz, 1008MHz, 1200MHz)	• 300 times 300×300 size matrices multiplication	• Scaling hotplug 300MHz to 1200 MHz
• Number of Cores (1 and 2)	• 300 times 300×300 size matrices multiplication	• Scaling interactive 300MHz to 1200MHz

an important consideration during job allocation of different mobile devices having heterogeneous resource availability.

D. RESULTS AND ANALYSIS

This subsection describes the experimental results in terms of execution time and energy obtained through varying the heterogeneity parameters (i.e., task size, workload, CPU speed, and number of cores). The details of tasks used to evaluate against individual parameters are listed in Table 2.

1) EXECUTION TIME

Fig. 1 shows the impact of heterogeneity parameters and random task allocation on execution time. Fig. 1 (a) clearly demonstrates that the execution time increases with the increased workload. This is mainly because of multitasking and context switching. The more the number of running applications, the fewer the processor cycles a task can get. During these experiments, we kept the size of the background running application (i.e. workload) constant and run it multiple times for better analysis. These results clearly suggest that ignoring the existing workload of MAC devices during task allocation can significantly prolong execution time, which may not be affordable for delay-sensitive applications.

Fig. 1 (b) shows the execution time of a running task on a single- and dual-core processor. To run a multi-threaded application, the Kernel Tuner application is used to customize the number of cores. It is evident from the figure that the execution time of a dual-core processor is about 37% less than the single, which is mainly because of parallelism. These results clearly indicate that the availability of more cores can significantly reduce task execution time. Therefore, an efficient task allocation strategy should consider the number of available cores during task allocation, e.g., compute-intensive tasks should be allocated to a multi-core processor.

The impact of processor speed on execution time is shown in Fig. 1 (c). We used master CPU android application to tune the CPU speed. The higher the processor speed, the shorter the execution time. Considering the fact that mobile devices may have heterogeneous processing capabilities, processor speed is therefore an important consideration during task allocation.

Fig. 1 (d) depicts the execution time as a function of task size. However, it is obvious from the figure that execution time increases with the increase in task size. The purpose is to demonstrate that MAC users might have variable size tasks and hence different execution time. Therefore, an efficient strategy must consider task size while choosing a mobile device for execution.

Fig. 1 (e) demonstrates the execution time (y-axis) of five tasks (x-axis), each of which was run on any of the four mobile devices (i.e., circles) picked randomly, which is represented through the large circles. As it is evident from the figure, random task allocation may lead to inappropriate node selection, which results in increased execution time. For example, randomly picked node 3 took 2250s to execute task 1, which could have been executed in 1500s at node 1. Similar findings can be observed for the other tasks (2-5).

2) ENERGY CONSUMPTION

Fig. 2 shows the impact of heterogeneity parameters on energy consumption. We used Power Tutor application to measure the energy consumption of a specified task. Fig. 2 (a) demonstrates the impact of varying workload on energy consumption. As the figure clearly indicates, energy consumption increases with the increase in workload. This is because a mobile device running more applications needs to perform extra computations, which requires more energy. Therefore, the current workload of a device is an important consideration while performing task allocation.

Fig. 2 (b) elucidates the energy consumed by a single and dual-core processor. A noticeable point is that a dual-core processor dissipates less energy than a single-core processor. This is mainly because the former exploits parallelism to process the same job quickly. These results advocate that

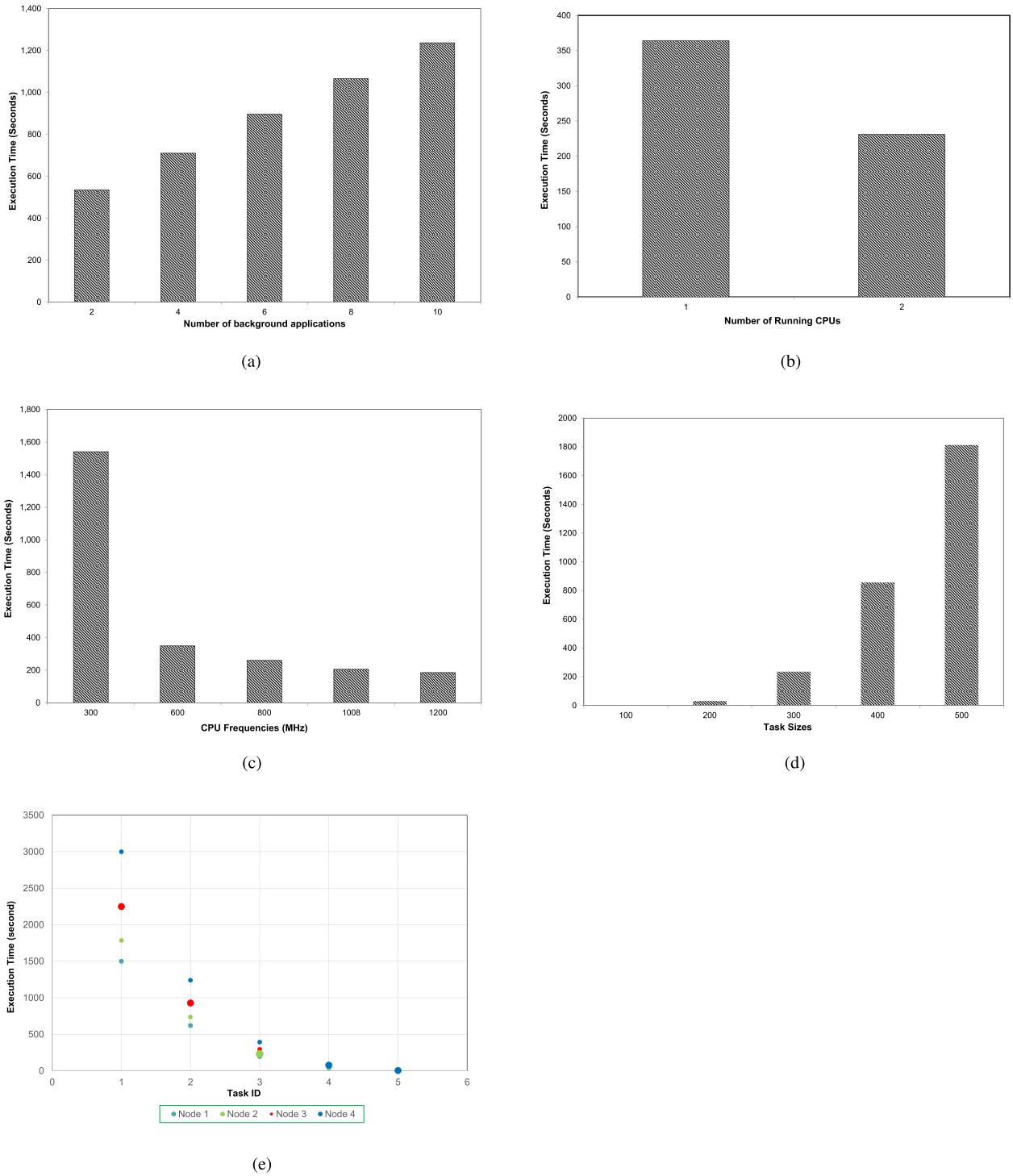

FIGURE 1. **Impact of heterogeneity parameters and random task allocation on execution time. (a) Impact of varying workload on execution time. (b) Execution time while varying number of cores. (c) Execution time as a function of processor speed. (d) Effect of varying task Size on execution time. (e) Impact of random-based task allocation on execution time.**

multi-core devices should be preferred for allocating a task to conserve energy.

Fig. 2 (c) shows the impact of processor speed (i.e., frequency) on energy consumption. These results clearly indicate that energy consumption increases with the increased processor frequencies. This is mainly because at lower frequencies a task is executed very slowly and its does not heat up the whole board of a device. To conserve energy, processor speed is an important consideration when performing task allocation. For example, a slower speed processor

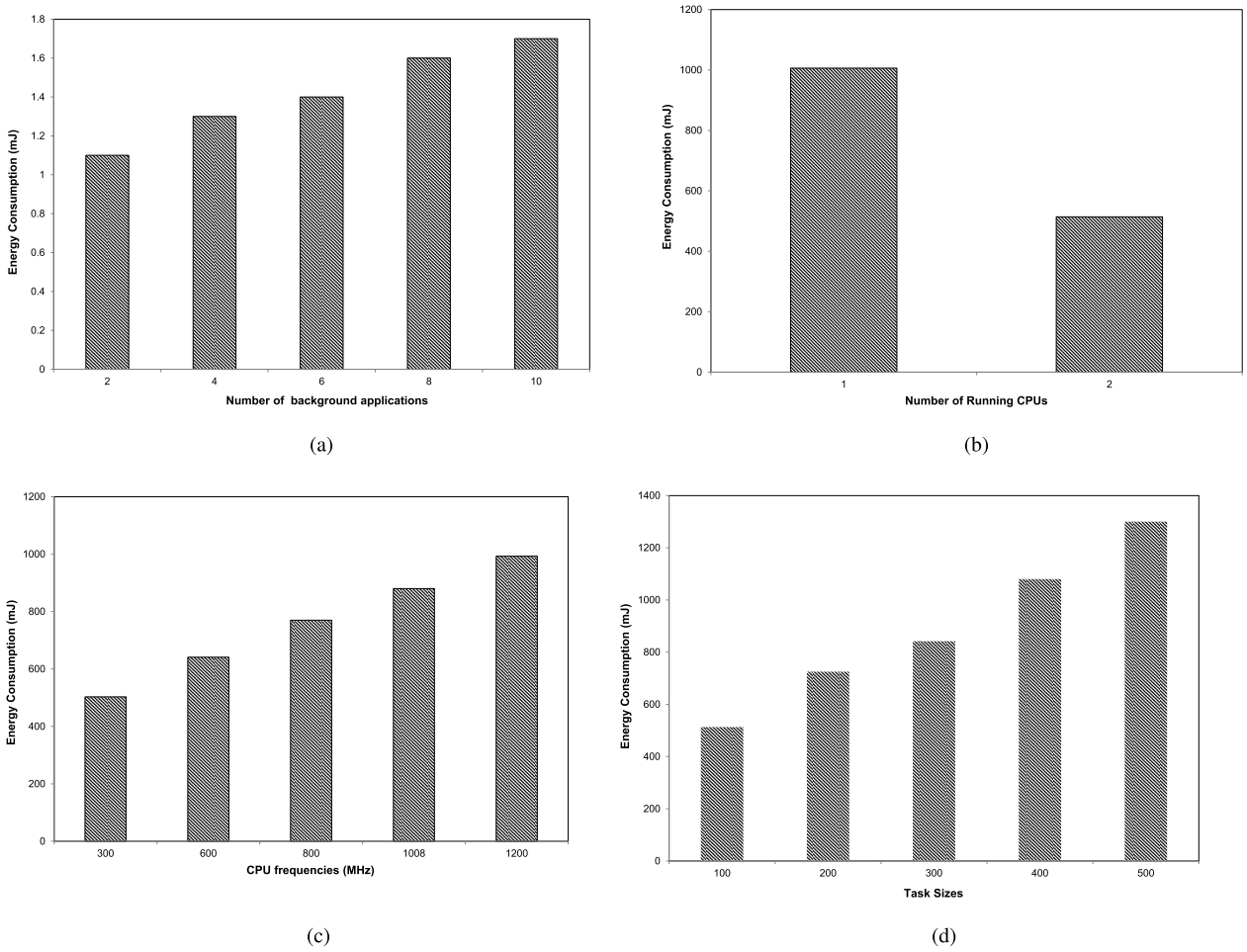

FIGURE 2. Impact of heterogeneity parameters on energy consumption. (a) Energy consumption as a function of workload. (b) Effect on energy consumption while changing number of cores. (c) Impact of varying processor speed on energy consumption. (d) Energy consumption as a function of varying task sizes.

might be allocated a task that does not require fast execution, such as in the case of a high speed processor with lower battery power.

Fig. 2 (d) depicts the effect of varying task size on energy consumption. As expected, the results indicate that the larger the task, the more energy it requires for execution since it involves more computations. Therefore, an effective MAC job allocation strategy must takes into account size of a task especially when devices may have heterogeneous energy resources.

This section clearly highlights the impact of ignoring resource and operational heterogeneity parameters on the performance of task execution in terms of time and energy. In the next section, we propose five heterogeneity-aware task allocation algorithms that take into account all these factors.

IV. HETEROGENEITY-AWARE TASK ALLOCATION ALGORITHMS

First, we describe a system model considered for this work. Then, we present five heterogeneity-aware task allocation algorithms for MAC.

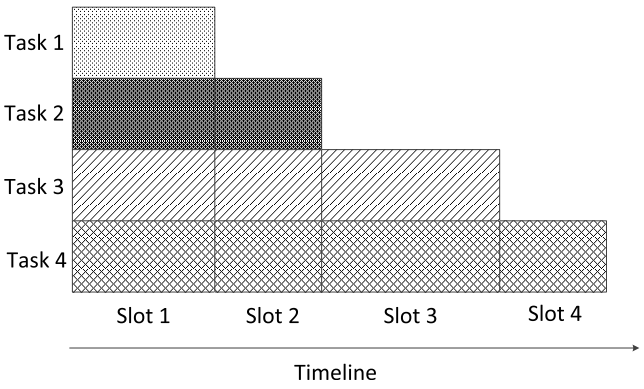

FIGURE 3. Tasks execution time illustration.

A. SYSTEM MODEL

To execute multiple tasks simultaneously, mobile devices implement time-sharing schemes to share CPU resources equally. Processor time slots are allocated to running tasks as shown in Fig. 3. We formulate equation 1 to calculate the task execution time. The slot time depends on the number of jobs

TABLE 3. Symbols description.

Symbols	Description
T_i	Execution time of i*th* task
$\mathbf{T_i}$	i*th* task
$I^{\mathcal{T}_s}$	Number of instructions of *i*th task executed in execution time slot *s*
$O^{\mathcal{T}_s}$	Number of instructions of background tasks executed in execution time slot *s*
P	Processor speed
T	Set of tasks
C	Number of cores
Val_{act}	Actual value
Val_{max}	Maximum value
Val_{min}	Minimum value
$\mathcal{T}_s$	Time of slot s
S_i	Size of i*th* slot
$norm(val_{act})$	Normalization of actual value

in execution i.e., it changes as a new task starts its execution or existing terminates. The aggregate execution of a task on a device may consist of several slots that vary with changes in the workload. The description of the used symbols is provided in Table 3.

$$T_i = \sum_{s=1}^{i} \mathcal{T}_s = \sum_{s=1}^{i} \left(\frac{I^{\mathcal{T}_s}}{(P \times C)} + \frac{O^{\mathcal{T}_s}}{(P \times C)}\right) \quad where\ i = 1 \ldots |T|\ and\ T_1 < T_2 \ldots < T_{|T|} \tag{1}$$

The execution time of an i*th* is the sum of all the slots taken by that task for the execution. The size of the first slot depends on the number of tasks being executed in that slot and the size of the smallest task. The size of the remaining time slots depends on the difference between the sizes of the slot numbered task and the next smaller task. The mathematical slot size can be modeled as presented in equation 2. To normalize the units of different variables, such as CPU speed, number of cores, and workload, equation 3 is used.

$$S_i = \begin{cases} (min(T) \times |T|)/(P \times C) \\ \text{if } (T_i - T_{i-1}) \times (|T| - (i+1))/(P \times C) \\ \text{if } i \geqslant 1 \end{cases} \tag{2}$$

$$norm(val_{act}) = [val_{act} - val_{min}] \times \left[\frac{1}{val_{max} - val_{min}}\right] \tag{3}$$

B. PROPOSED ALGORITHMS

As mentioned earlier in Section IV, processor speed is an important consideration while performing task allocation. Therefore, our proposed algorithm 1 allocates tasks based on CPU speed. The symbols N, T, and S_N are used as input parameters. (see Table 4 for the description of the symbols.) In case there is more than one task, this algorithm sorts them in descending order based on their length (line 2). The controller prefers to pick a mobile device based on the fastest processor speed, as this will result in faster execution and hence less energy consumption (lines 4-9). Although

Algorithm 1 CPU Speed-Based Task Allocation Algorithm

Input: N, T, S_N
1 $X \leftarrow T$
2 Sort(Desend, X)
3 **for** i=1:|X| **do**
4 $(\widehat{s_n}) = arg\ max_{\forall\, n \in N\ \forall\, s_n \in S_N} f(s_n)$
5 NodeID $\leftarrow getID(N, \widehat{s_n})$
6 map<NodeID> $\leftarrow x_{\{1\}}\ where\ x_{\{1\}} \in x$
7 X $\leftarrow X/x_1$
8 **end**
9 **OUTPUT**:map<N,X>

TABLE 4. Description of the symbols used in the algorithms.

Symbol	Description
N	Number of mobile devices
T	Number of given tasks or sub-tasks
S_N	CPU speed of N mobile device that is computed by multiplying it with number of cores
C_N	Number of cores of N mobile device
$\mathbf{C_N}$	Total capacity of N mobile device
R_T	Real execution time of the task
$\mathbf{R_N}$	Residual capacity of N mobile device in terms of workload
CPI	Cycle per instruction of specified CPU architecture of mobile device
E^n_{tL}	Expected workload on n*th* node (because of the task execution)
$\widehat{\ }$	All symbols with$\widehat{\ }$represent the maximum or minimum value returned by the function (arg)

selection of such a device will result in less execution time, further improvement is possible with the consideration of other parameters as we shall discuss later in this section.

Algorithm 2 Core-Based Task Allocation Algorithm

Input: N, T, C_N
1 X $\leftarrow T$
2 Sort(Desend, X)
3 **for** i=1:|X| **do**
4 $(\widehat{c_n}) = arg\ max_{\forall\, n \in N\ \forall\, c_n \in C_N} f(c_n)$
5 NodeID $\leftarrow getID(N, \widehat{c_n})$
6 map<NodeID> $\leftarrow x_{\{1\}}\ where\ x_{\{1\}} \in x$
7 X $\leftarrow X/x_1$
8 **end**
9 **OUTPUT**:map<N,X>

Algorithm 2 takes into account the number of cores while performing task allocation. Again, the tasks are sorted in descending order (line 2). As the controller node maintains the updated information of the mobile devices participating in MAC, it prefers to pick a device with the highest number of available cores to execute a task. (lines 4-7). This is mainly because multi-threaded tasks may execute in parallel and results in less execution time. Moreover, considering other parameters, such as workload, task size, and processor speed can further reduce execution time.

Algorithm 3 Workload-Based Task Allocation Algorithm

Input: $N, T, CPI, \mathbf{C_N}, R_T, \mathbf{R_N}$
1 $X \leftarrow T$
2 Sort(Desend, X)
3 $\forall_{n=1..|N|} E^n_{tL} = 0$
4 **for** i=1:|X| **do**
5 $\forall n \in N \; map < X_i, E^n_L > \leftarrow \dfrac{|X_i| \times \frac{1}{S_n} \times CPI_n}{R_{X^n_i}} \times 100$
6 **end**
7 $\forall n \in N, \; \mathbf{R_n} = \mathbf{C_n} - E^n_{tL}$
8 **for** i=1:|X| **do**
9 while(|X| != NULL)
10 $(\widehat{w_n}) = arg\ max_{\forall n \in N, w_n \in \mathbf{R_N}} f(w_n)$
11 $f(w_n) \leftarrow w_n$
12 NodeID $\leftarrow getID(N, \hat{w}_n)$
13 map<NodeID> $\leftarrow x_{\{1\}} where\ x_{\{1\}} \in X$
14 $E^n_{tL} \leftarrow E^n_{tL} + E^n_L$;
15 $X \leftarrow X/x_1$
16 **end**
17 **OUTPUT**:map<N,X>

Algorithm 4 Two Parameters-Based (CPU Speed and Workload) Task Allocation Algorithm

Input: $N, S_N, T, \mathbf{C_N}, CPI, R_T, \mathbf{R_N}$
1 $X \leftarrow T$
2 Sort(Desend, X)
3 $\forall_{n=1..|N|} E^n_{tL} = 0$
4 **for** i=1:|X| **do**
5 $\forall n \in N \; map < X_i, E^n_L > \leftarrow \dfrac{|X_i| \times \frac{1}{S_n} \times CPI_n}{R_{X^n_i}} \times 100$
6 **end**
7 $\forall n \in N, \; \mathbf{R_n} = \mathbf{C_n} - E^n_{tL}$
8 while(|X| != NULL)
9 $(\widehat{w_n}, \widehat{s_n}) = arg\ max_{\forall n \in N, w_n \in \mathbf{R_N}, s_n \in S_N} f(w_n, s_n)$
10 f(w_n, s_n) $\leftarrow \alpha \times w_n + \beta \times s_n$
11 NodeID $\leftarrow getID(N, \hat{w}_n, \hat{c}_n)$
12 map<NodeID> $\leftarrow x_{\{1\}} where\ x_{\{1\}} \in X$
13 $E^n_{tL} \leftarrow E^n_{tL} + E^n_L$;
14 $X \leftarrow X/x_1$
15 **end**
16 **OUTPUT**:map<N,X>

As stated earlier, various MAC devices may have different workload. In such as a situation, inflicting a particular device may not only increase execution time but also dissipates more energy. Therefore, Algorithm 3 allocates task based on workload. First, the tasks are sorted based on their size in descending order (line 2). Then, the controller node determines the residual capacity $\mathbf{R_N}$ of each MAC device in real-time by subtracting its current workload from the total capacity $\mathbf{C_N}$ (line 7). The tasks are allocated to mobile devices based on their residual capacity. Although this solution helps to minimize execution time, further improvement is also possible. For example, choosing a device with low workload may not help much because of the slow processing speed. Therefore, we present another task allocation algorithm based on processor speed and workload in the following.

Algorithm 4 is designed to allocate tasks based on processor speed and workload. The description of the used symbols is presented in Table 4. Tasks to be allocated are sorted in descending order first (line 2). In order to find the residual workload, the expected workload of each task to be executed on the compute node is calculated (lines 5-7). A weighted graph formula is derived to incorporate both the two parameters (i.e., CPU speed and workload) while performing task allocation (lines 9-11). Tasks are allocated based on the maximum values derived from the weighted average formula. A noticeable point is that depending upon the significance of both the parameters, the corresponding weights can be tuned and adjusted accordingly. For example, we have selected and assigned 0.05 and 0.95 weights to processor speed and workload, respectively. Algorithm 4 takes into account processor speed and workload but does not consider the number of cores, which is an important design consideration as discussed earlier in Section III (D). Therefore, we present a variant of algorithm 4 in the following, which also considers the number of cores.

While performing task allocation, algorithm 5 takes into account resource and operational heterogeneity (i.e., processor speed, number of cores, and workload). The reader is referred to Table 4 for a description of the notations. Once the controller device receives a set of tasks to be executed, it sorts them in descending order (lines 1-3). The workload on each compute node is calculated (line 5). Since there is more than one parameter, weights are assigned to each of them based on their significance, i.e., CPU speed (0.15), cores (0.20), and workload (0.65). These weights can be tuned and adjusted. Based on the average weighted formula, devices are selected and tasks are assigned (lines 5-12) and workload information is updated to avoid inconsistency (lines 13-14).

V. EXPERIMENTAL EVALUATION

Extensive simulation experiments are performed to evaluate the effectiveness of the proposed algorithms. This section describes the experiment setup, performance metrics, and analysis of results.

A. EXPERIMENT SETUP AND PERFORMANCE METRICS

The proposed heterogeneity-aware task allocation algorithms for MAC are implemented in MATLAB. To conduct experiments, one controller and four mobile devices with different specifications (Table 5) are simulated. The proposed task allocation algorithms are implemented on a controller node

Algorithm 5 Three Parameters-based (CPU Speed, Workload, and Core) Task Allocation Algorithm

Input: $N, S_N, T, C_N, \mathbf{C_N}, CPI, R_T, \mathbf{R_N}$

$X \leftarrow T$

Sort(Desend, X)

$\forall_{n=1..|N|}\ E^n_{tL} = 0$ **for** i=1:|X| **do**

$$\forall n \in N\ map < X_i, E^n_L > \leftarrow \frac{|X_i| \times \frac{1}{S_n} \times CPI_n}{R_{X^n_i}} \times 100$$

end

$\forall n \in N, \mathbf{R_n} = \mathbf{C_n} - E^n_{tL}$

while(|X| != NULL)

$(\widehat{w_n}, \widehat{s_n}, \widehat{c_n}) = arg\ max_{\forall\, n \in N,\ w_n \in \mathbf{R_N}}\ s_n \in S_N,\ c_n \in C_N,\ f(w_n, s_n, c_n)$

$\mathrm{f}(w_n, s_n, c_n) \leftarrow \alpha \times w_n + \beta \times s_n + \gamma \times c_n$

NodeID $\leftarrow getID(N, \hat{w_n}, \hat{s_n}, \hat{c_n})$

map<NodeID> $\leftarrow x_{\{1\}}\ where\ x_{\{1\}} \in X$

$\mathrm{E}^n_{tL} \leftarrow E^n_{tL} + E^n_L$;

$X \leftarrow X/x_1$

end

OUTPUT:map<N,X>

TABLE 5. Specification of mobile device used in simulation.

Mobile	Processor Speed	Number of Core
Mobile 1	1300MIPS	Quad-core
Mobile 2	1008MIPS	Single-core
Mobile 3	800MIPS	Dual-core
Mobile 4	600MIPS	Octa-core

for job distribution to compute devices. Simulation configuration was verified through results comparison with real mobile devices, e.g., the same processing speed (i.e., million instructions per second (MIPS)) as real mobile devices.

The same multi-threaded matrix multiplication application discussed earlier in Section III (B) was run. In simulation experiments, we used Lackey (Valgrind tool) to calculate the number of instructions of the application. Table 6 presents the computational lengths of the tasks that were used for the experiments (one task is comprised of five sub-tasks). We used the same performance metrics (i.e., execution time and energy consumption) as described earlier in Section III (B).

B. RESULTS AND ANALYSIS

We compare the performance of the proposed algorithms to that of random-based (RM) task allocation in terms of execution time and energy. As described earlier, the proposed algorithms are based on processor speed (SO), number of cores (CO), workload (WO), speed and workload (SW), and speed, number of cores, and workload (SCW). The results are based on 30 data traces. The execution time is measured in seconds.

1) EXECUTION TIME (SECONDS)

Fig. 4 shows the execution time of 30 data traces, measured by running the SO- and RM-based task allocation. The results

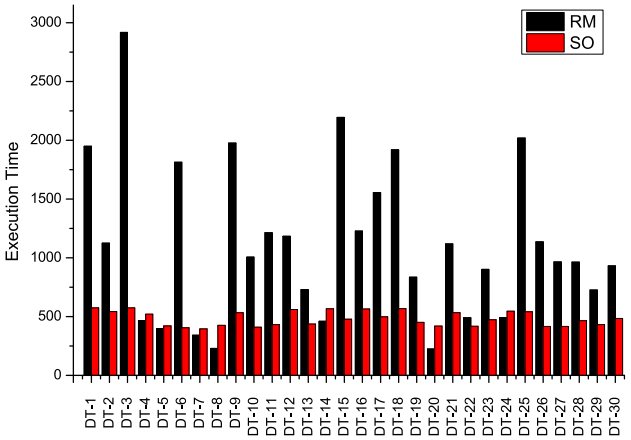

FIGURE 4. Execution time of only CPU speed based-task allocation.

indicate a significant reduction in execution time compared with the random-based solution in most of the cases. The reason for the reduction in execution time is that through a proposed solution, the task assignment decision is made based on CPU speed. In this context, the tasks are sorted in ascending order and then assigned to the compute nodes on the basis of the CPU speed. Thus, the allocation of the larger task on high-speed mobile devices helps to minimize the execution time. However, in random-based solution, most of the time this task is assigned to a device that has low processing capabilities. In the simulation scenario this was found to cause inefficient resource utilization, which can degrade task performance by delaying the execution time. In addition, inefficient resource utilization can increase energy consumption in the MAC environment.

Fig. 5 depicts the execution time of 30 data traces obtained from CO- and RM- based task allocation. The results indicate that the proposed solution helps to execute most of the data traces in a fast manner as their execution time was found to be much lower compared with the random-based mechanism. In random-based task allocation, the controller

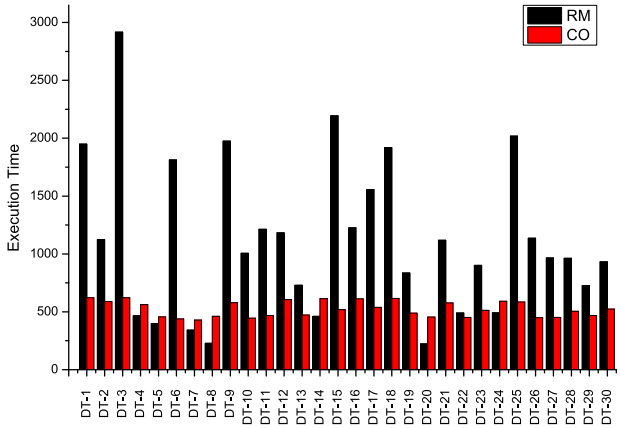

FIGURE 5. Execution time of core-based task allocation.

TABLE 6. Data traces for evaluations of various parameters.

Data Trace #	Computational Lengths of Five Sub-tasks (equal to one task or data trace)				
Data Trace-1	1.0e+12 × 0.0016	1.0e+12 × 0.0419	1.0e+12 × 0.2355	1.0e+12 × 0.7434	1.0e+12 × 1.9649
Data Trace-2	1.0e+12 × 0.0017	1.0e+12 × 0.0468	1.0e+12 × 0.2396	1.0e+12 × 0.7368	1.0e+12 × 1.8003
Data Trace-3	1.0e+12 × 0.0016	1.0e+12 × 0.0430	1.0e+12 × 0.2392	1.0e+12 × 0.7411	1.0e+12 × 1.9595
Data Trace-4	1.0e+12 × 0.0024	1.0e+12 × 0.0402	1.0e+12 × 0.2385	1.0e+12 × 0.7431	1.0e+12 × 1.6787
Data Trace-5	1.0e+12 × 0.0026	1.0e+12 × 0.0452	1.0e+12 × 0.2339	1.0e+12 × 0.7392	1.0e+12 × 1.1712
Data Trace-6	1.0e+12 × 0.0025	1.0e+12 × 0.0402	1.0e+12 × 0.2328	1.0e+12 × 0.7306	1.0e+12 × 1.0971
Data Trace-7	1.0e+12 × 0.0027	1.0e+12 × 0.0449	1.0e+12 × 0.2332	1.0e+12 × 0.7433	1.0e+12 × 1.0344
Data Trace-8	1.0e+12 × 0.0021	1.0e+12 × 0.0427	1.0e+12 × 0.2377	1.0e+12 × 0.7411	1.0e+12 × 1.1869
Data Trace-9	1.0e+12 × 0.0022	1.0e+12 × 0.0431	1.0e+12 × 0.2365	1.0e+12 × 0.7399	1.0e+12 × 1.7547
Data Trace-10	1.0e+12 × 0.0018	1.0e+12 × 0.0448	1.0e+12 × 0.2366	1.0e+12 × 0.7323	1.0e+12 × 1.1190
Data Trace-11	1.0e+12 × 0.0022	1.0e+12 × 0.0467	1.0e+12 × 0.2334	1.0e+12 × 0.7382	1.0e+12 × 1.2238
Data Trace-12	1.0e+12 × 0.0026	1.0e+12 × 0.0418	1.0e+12 × 0.2351	1.0e+12 × 0.7398	1.0e+12 × 1.8909
Data Trace-13	1.0e+12 × 0.0029	1.0e+12 × 0.0438	1.0e+12 × 0.2314	1.0e+12 × 0.7321	1.0e+12 × 1.2575
Data Trace-14	1.0e+12 × 0.0027	1.0e+12 × 0.0418	1.0e+12 × 0.2381	1.0e+12 × 0.7334	1.0e+12 × 1.9293
Data Trace-15	1.0e+12 × 0.0020	1.0e+12 × 0.0414	1.0e+12 × 0.2325	1.0e+12 × 0.7386	1.0e+12 × 1.4733
Data Trace-16	1.0e+12 × 0.0020	1.0e+12 × 0.0458	1.0e+12 × 0.2359	1.0e+12 × 0.7377	1.0e+12 × 1.9172
Data Trace-17	1.0e+12 × 0.0019	1.0e+12 × 0.0453	1.0e+12 × 0.2375	1.0e+12 × 0.7353	1.0e+12 × 1.5678
Data Trace-18	1.0e+12 × 0.0015	1.0e+12 × 0.0404	1.0e+12 × 0.2353	1.0e+12 × 0.7409	1.0e+12 × 1.9340
Data Trace-19	1.0e+12 × 0.0016	1.0e+12 × 0.0440	1.0e+12 × 0.2347	1.0e+12 × 0.7302	1.0e+12 × 1.3371
Data Trace-20	1.0e+12 × 0.0017	1.0e+12 × 0.0456	1.0e+12 × 0.2331	1.0e+12 × 0.7374	1.0e+12 × 1.1656
Data Trace-21	1.0e+12 × 0.0024	1.0e+12 × 0.0418	1.0e+12 × 0.2365	1.0e+12 × 0.7396	1.0e+12 × 1.7482
Data Trace-22	1.0e+12 × 0.0021	1.0e+12 × 0.0406	1.0e+12 × 0.2323	1.0e+12 × 0.7428	1.0e+12 × 1.1524
Data Trace-23	1.0e+12 × 0.0027	1.0e+12 × 0.0438	1.0e+12 × 0.2400	1.0e+12 × 0.7311	1.0e+12 × 1.4427
Data Trace-24	1.0e+12 × 0.0016	1.0e+12 × 0.0467	1.0e+12 × 0.2300	1.0e+12 × 0.7408	1.0e+12 × 1.8173
Data Trace-25	1.0e+12 × 0.0028	1.0e+12 × 0.0406	1.0e+12 × 0.2340	1.0e+12 × 0.7336	1.0e+12 × 1.8001
Data Trace-26	1.0e+12 × 0.0021	1.0e+12 × 0.0464	1.0e+12 × 0.2318	1.0e+12 × 0.7337	1.0e+12 × 1.1455
Data Trace-27	1.0e+12 × 0.0016	1.0e+12 × 0.0461	1.0e+12 × 0.2358	1.0e+12 × 0.7377	1.0e+12 × 1.1450
Data Trace-28	1.0e+12 × 0.0028	1.0e+12 × 0.0444	1.0e+12 × 0.2335	1.0e+12 × 0.7372	1.0e+12 × 1.4018
Data Trace-29	1.0e+12 × 0.0015	1.0e+12 × 0.0417	1.0e+12 × 0.2312	1.0e+12 × 0.7326	1.0e+12 × 1.2400
Data Trace-30	1.0e+12 × 0.0021	1.0e+12 × 0.0403	1.0e+12 × 0.2390	1.0e+12 × 0.7432	1.0e+12 × 1.4909

node assigns tasks to the compute nodes without considering the specification of the compute nodes as noticed in the simulation scenario. Therefore, random-based task allocation results in delaying the execution time in most of the data traces execution. Although using core-based solution execution time can be minimized, further reduction in execution time is possible if other parameters, such as workload and CPU speed, are incorporated. In addition, multiple parameter-based task allocation solutions can help to obtain an optimal reduction in execution time.

Fig. 6 presents the execution time of 30 data traces that are measured through WO- and RM-based task allocation. The results reveal that workload-based task allocation helps to execute most of the data traces in a fast manner compared with the random-based mechanism. In our simulation scenario, it has been noticed that in random-based task allocation, the controller node assigns tasks to compute nodes that have lower processing capabilities. Therefore, random-based task allocation prolongs the execution time. However, through the workload-based task allocation, compute nodes were being selected in the simulation scenario that have high specifications, as they have a lighter work-load running in the background. Although the use of a workload-based solution can minimize execution time, further reduction in execution time is also possible if the task allocation decision is based on two or three parameters instead of just one.

Fig. 7 shows the execution time of the 30 data traces obtained through SW- and RM-based task allocation.

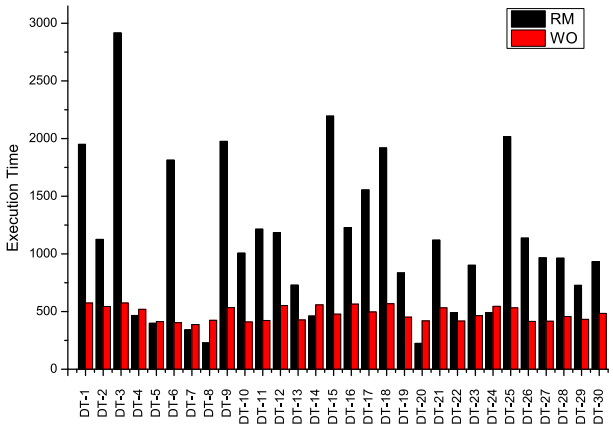

FIGURE 6. Execution time of only workload-based task allocation.

The results reveal that the execution time measured through two parameters-based task allocation is much lower compared with a random-based solution in most of the cases. The simulation scenario reveals that through random-based task allocation, most of the time the tasks were allocated to slower devices for the execution. Thus, the tasks took longer to complete. However, the proposed solution ensures minimization in task execution time due to the involvement of two parameters (CPU speed and workload) when the task allocation decision was made. In our simulation scenario, through the proposed solution devices that have high

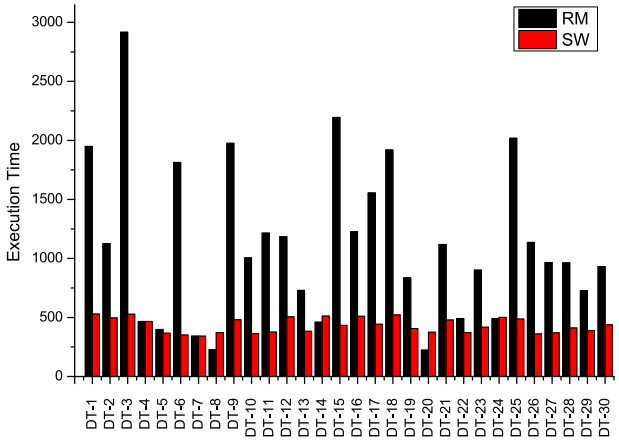

FIGURE 7. Execution time of two parameters- (CPU speed and workload) based task allocation.

specifications which ensure minimization in execution time are selected. Although performing task allocation based on two parameters helps to minimize the execution time compared with single parameter-based task allocation, further reduction in execution time is also possible if the task allocation decision is based on three parameters.

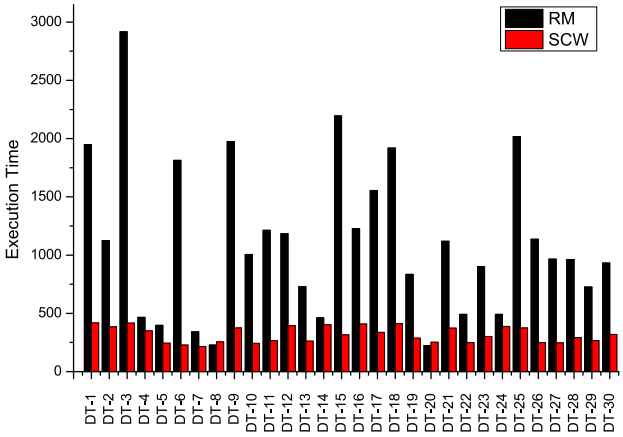

FIGURE 8. Execution time of two parameters- (CPU speed, number of cores, and coreworkload) based task allocation.

Fig. 8 illustrates the execution time of 30 data traces obtained through SCW- and RM-based task allocation. The results show that the proposed solution executes 30 data traces generally in less time than random-based task allocation. The reason is that in the simulation scenario, larger tasks were almost always allocated to devices that have lower processing capabilities. However, through a three parameters-based task allocation, compute nodes that have high processing capabilities were selected in our simulation scenario. Therefore, the proposed solution outperforms the random-based task allocation in most of the data traces execution in terms of execution time. Thus, it is concluded that performing task allocation based on CPU speed, core, and workload running in the background can lead to efficient resource utilization in MAC. This efficient resource utilization minimizes the execution time of the tasks more than the random and other four proposed solutions.

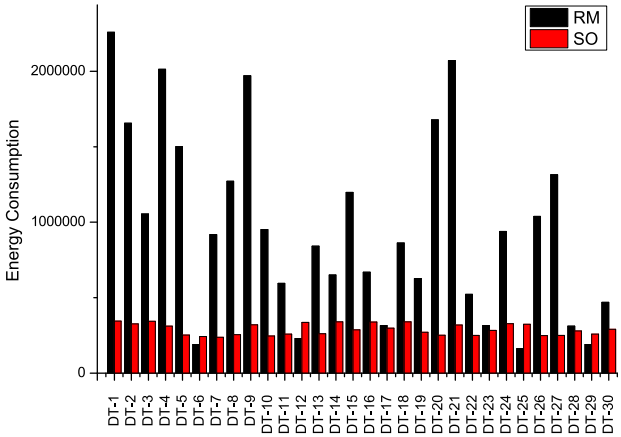

FIGURE 9. Energy consumption measured through CPU speed-based task allocation.

2) ENERGY CONSUMPTION (mJ)

Fig. 9 presents the results of 30 data traces obtained from SO- and RM-based task allocation. The results reveal that random-based task allocation consumes more energy compared with CPU speed-based task allocation in most of the cases. The reason is that the random-based task allocation performs task assignment to slower devices in most of the cases, as it has been noticed in the simulation scenario. The allocation of tasks to slower devices leads to energy wastage. However, CPU speed-based task allocation enables the controller node to assign all the tasks only to devices that have high CPU speed, which helps to execute a task by consuming less energy.

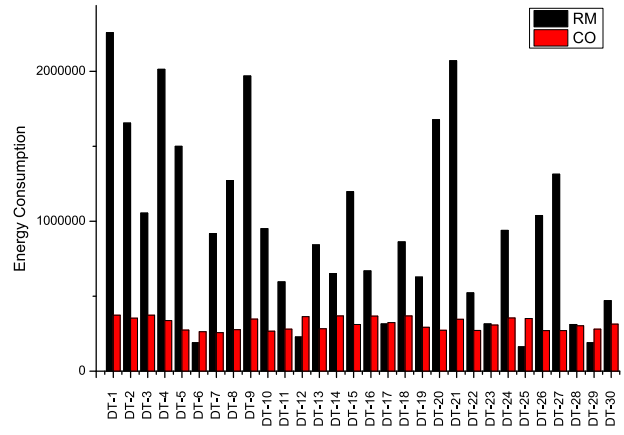

FIGURE 10. Energy consumption measured through core-based task allocation.

Fig. 10 presents the results of 30 data traces that are obtained from the CO- and RM-based task allocation. The results reveal that random-based task allocation usually

consumes more energy due to allocating tasks to slower devices in most of the cases, as observed in the simulation scenario. However, core-based task allocation enables the controller node to select only the devices for task execution that have the most cores. Thus, core-based task allocation helps to select such devices for computation that have high processing capabilities. In this way, this solution not only minimizes the execution time but also saves energy compared with random-based task allocation.

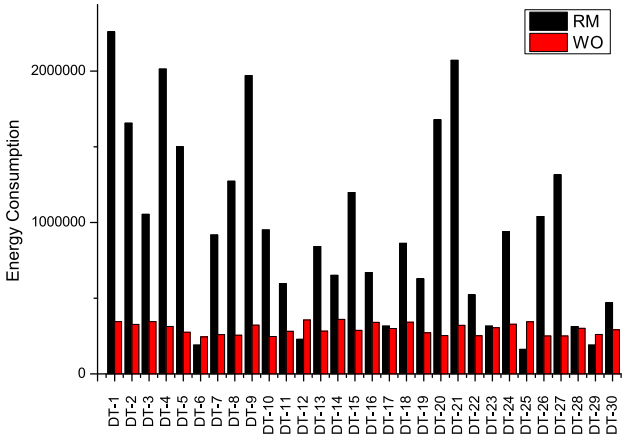

FIGURE 11. Energy consumption measured through workload-based task allocation.

Fig. 11 illustrates the results of energy consumption obtained from WO- and RM-based task allocation. The results clearly show that random-based task allocation usually consumes more energy for executing different data traces. However, workload-based task allocation consumes less. The reason is that, particularly in our simulation scenario, the devices selected based on workload criteria have high processing capabilities. Therefore, this solution consumes less energy than random-based task allocation.

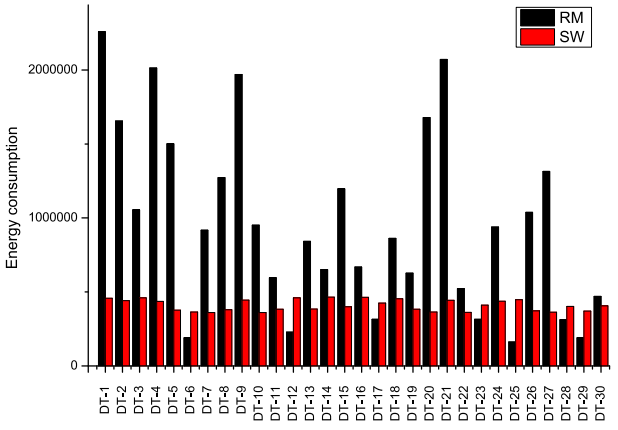

FIGURE 12. Energy consumption measured through two parameters-(CPU speed and workload) based task allocation.

Fig. 12 presents the results obtained from SW- and RM-based task allocation. The results show that the proposed solution outperforms the random-based task allocation. The graph shows that most of the data traces are consuming more energy through random-based task allocation in most of the cases. The reason is that in our simulation scenario, it was observed that through random-based task allocation, the controller node usually assigns larger tasks in terms of computational lengths to slower devices that lead to the consumption of extra energy. However, the compute nodes selected through the proposed solution had high processing capabilities. Therefore, the proposed solution consumes less energy than random-based task allocation.

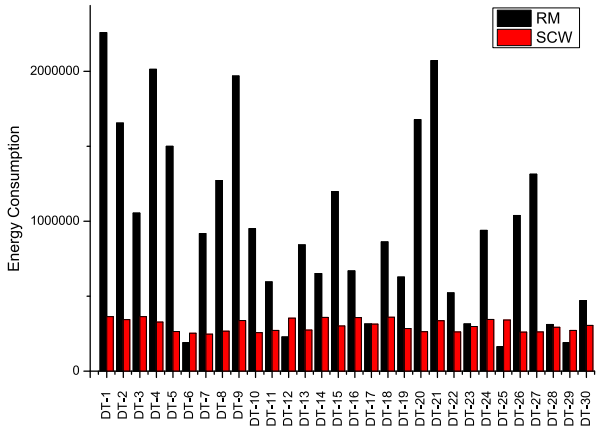

FIGURE 13. Energy consumption measured through three parameters-(CPU speed, number of cores, and workload) based task allocation.

Fig. 13 presents the results obtained from SCW- and RM-based task allocation. The results reveal that random-based task allocation results in the consumption of more energy than the proposed solution in most cases. The reason is that through random-based task allocation, tasks are being assigned to the slower devices as noted in the simulation scenario. However, the proposed solution ensures that tasks are allocated to devices that have high CPU speed, core, and less workload. Based on this defined task allocation criteria, the devices that have high specifications were being selected for task execution in our simulation scenario. Thus, such selection is better at ensuring minimum energy consumption than random-based task allocation.

VI. SYSTEM MODEL VALIDATION

The correctness of the developed system model is validated by comparing the results obtained from a system model with that of simulation. The execution time is used as a parameter to validate the system model.

A. EXECUTION TIME OF FIVE HETEROGENEITY-AWARE TASK ALLOCATION SOLUTIONS

1) SO VS. RM

Fig. 14 shows the comparison of execution time measured through the simulation of a CPU speed-based solution with the system model results. The X and Y axes show five data

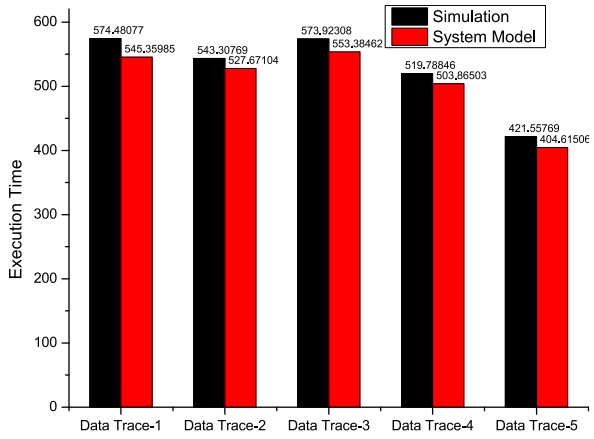

FIGURE 14. Execution time of CPU speed-based task allocation.

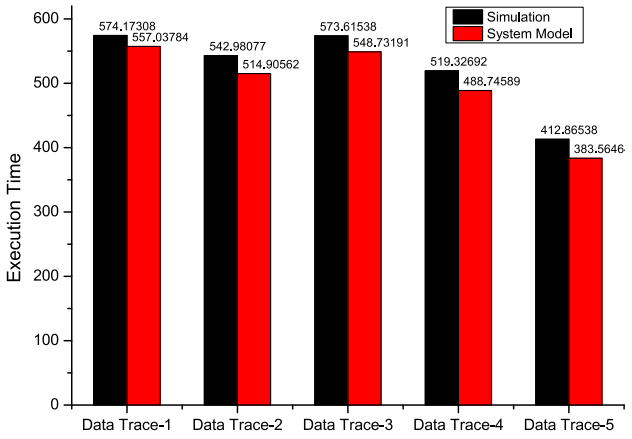

FIGURE 16. Execution time of workload-based task allocation.

traces and execution times (in seconds), respectively. The graph shows that the experimental results of five data traces are closer to the results obtained from the system model. The differences in execution time through simulation results and system models are measured at 5.06% of data trace 1, 2.87% of data trace 2, 3.57% of data trace 3, 3.06% of data trace 4, and 4.01% of data trace 5.

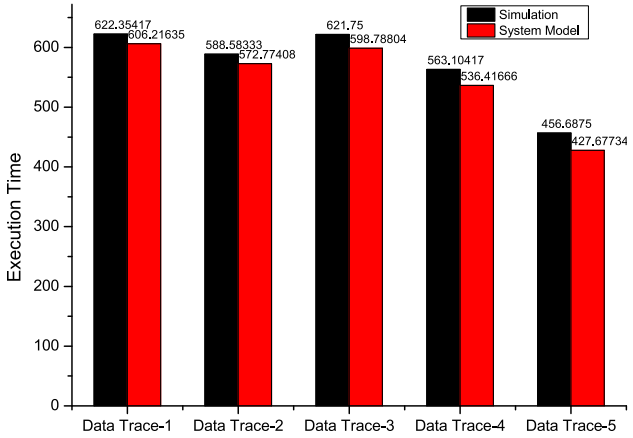

FIGURE 15. Execution time of core-based task allocation.

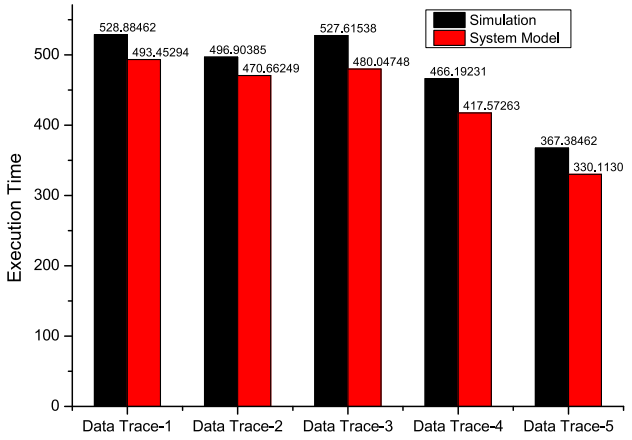

FIGURE 17. Execution time of two parameters- (CPU speed and workload) based task allocation.

2) CO VS. RM

Fig. 15 shows the comparison of execution time measured through the simulation of a core-based solution with the system model results. The X and Y axes show five data traces and execution times (in seconds), respectively. The graph shows that the experimental results of five data traces are closer to the results obtained from the system model. The differences in execution time through simulation results and system models are measured at 2.59% of data trace 1, 2.68% of data trace 2, 3.69% of data trace 3, 4.73% of data trace 4, and 6.35% of data trace 5.

3) WO VS. RM

Fig. 16 shows the comparison of execution time measured through the simulation of a workload-based solution with the system model results. The X and Y axes show five data traces and execution times (in seconds), respectively. The graph shows that the experimental results of five data traces are closer to the results obtained from the system model. The differences in execution time through simulation results and system models are measured at 2.98% of data trace 1, 5.17% of data trace 2, 4.33% of data trace 3, 5.88% of data trace 4, and 7.09% of data trace 5.

4) SW VS. RM

Fig. 17 shows the comparison of execution time measured through the simulation of two parameters (CPU speed and workload) based on a solution with the system model results. The X and Y axes show five data traces and execution times (in seconds), respectively. The graph shows that the experimental results of five data traces are closer to the results obtained from the system model. The differences in execution time through simulation results and system models are measured at 6.69% of data trace 1, 5.28% of data trace 2, 9.01% of data trace 3, 10.42% of data trace 4, and 10.14% of data trace 5.

5) SCW VS. RM

Fig. 18 shows the comparison of execution time measured through simulation of three parameters

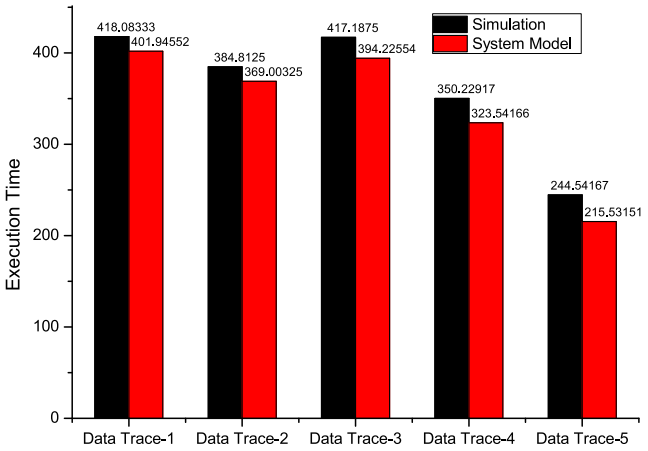

FIGURE 18. Execution time of three parameters- (CPU speed, number of cores, and workload) based task allocation.

(CPU speed, core, and workload) based on a solution with the system model results. The X and Y axes show five data traces and execution times (in seconds), respectively. The graph shows that the experimental results of five data traces are closer to the results obtained from the system model. The differences in execution time through simulation results and system models are measured at 3.85% of data trace 1, 4.10% of data trace 2, 5.50% of data trace 3, 7.62% of data trace 4, and 11.86% of data trace 5.

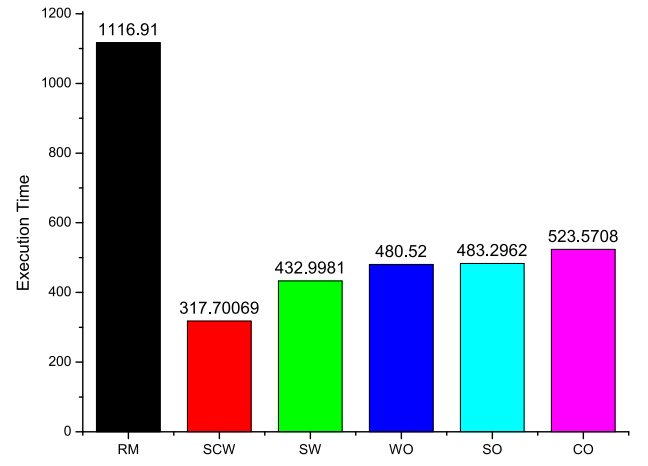

FIGURE 19. Comparison of execution time empirical results obtained from five proposed solutions with random-based task allocation.

VII. MEAN VALUES-BASED COMPARISON WITH RANDOM-BASED TASK ALLOCATION IN TERMS OF EXECUTION TIME

Fig. 19 presents the execution time of five tasks. The x and y axes represent different types of proposed algorithms and execution time, respectively. The execution time is measured in seconds. The comparison results reveal that proposed solutions outperform the RM-based task allocation.

In RM-based task allocation, tasks of various sizes are assigned to the available mobile devices without considering resources and operational heterogeneity in the MAC paradigm. This causes inefficient resource utilization that can prolong the execution time of the task. In this context, five task allocation solutions are proposed. One of the solutions is to base the decision on CPU speed while performing task allocation. The execution of the tasks that are performed based on high CPU speed mobile devices helps to shorten the task execution time compared with low CPU speed mobile devices that can be selected by random-based task allocation. Thus, the results of the solution based on high CPU speed show that it can shorten the task execution time by as much as 56.72% compared with random-based task allocation.

The second possible solution is based on the number of cores. First, it needs to be made clear that a core-based solution is different from a CPU speed-based task allocation. In the experiment, the devices were selected so that high-speed CPU devices do not have a higher number of cores, otherwise the result of the high CPU speed and the core-based proposed solution could remain the same. Thus, the results of the core-based proposed solution indicate that performing task allocations based on the higher number of cores helps to minimize the execution time by up to 53.12% compared with random-based task allocation. The reason for the shorter execution time is that the tasks that required more computing power were being executed on resource-rich mobile devices with a higher number of cores. The device with a higher number of cores ensures parallel processing of multi-threaded tasks.

The third proposed solution is based on the workload parameter. In MAC, the devices may have a different workload running in the background. Therefore, basing task allocation only on high CPU speed and a larger number of cores may not be useful when the workload on high-speed mobile devices is very high. In such cases, basing task allocation on high CPU speed or number of cores can degrade the performance of the tasks in comparison with devices that have low CPU speed and fewer cores; however, the background workload on the devices is very low. In this context, workload-based task allocation can minimize the task execution time. Thus, the workload-based solution can improve the task execution by up to 56.97% compared with random-based task allocation.

Although performing task allocation based on a single parameter helps to minimize the execution time, a greater reduction in execution time is possible by selecting compute nodes based on two and three parameters. Moreover, the solution of task allocation by considering only one parameter does not guarantee optimal reduction in the task execution time. In order to see a shorter execution time, the nodes for performing task execution must be selected based on a simultaneous high CPU speed and a lighter running workload. This ensures improvement in task execution time up to 61.23% compared with a random-based solution.

Although a significant reduction in execution time is seen in these four solutions, further reduction is also possible by basing task allocation on three parameters together, such as CPU speed, number of cores, and workload.

The consideration of these parameters together leads to more appropriate resource utilization than a random-based task allocation solution. Efficient resource utilization enables the mobile devices to execute the task quickly. Thus, the results show a substantial increase in the performance by reducing the execution time by up to 71.55% through a final proposed task allocation solution (SCW) that is based on three parameters: CPU speed, workload, and number of cores. In addition, the three parameters-based solution is analyzed as one of the best of the five in terms of execution time in our scenario.

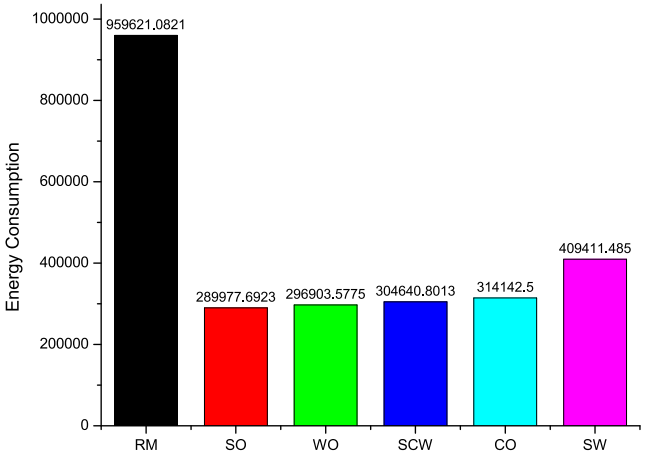

FIGURE 20. Comparison of energy consumption empirical results obtained from five proposed solutions with random-based task allocation.

VIII. MEAN VALUES-BASED COMPARISON WITH RANDOM-BASED TASK ALLOCATION IN TERMS OF ENERGY CONSUMPTION

Fig. 20 presents the results of energy consumption that are obtained through proposed heterogeneity-aware task allocation solutions in our simulated scenario. The purpose of this section is to discuss the performance of proposed solutions in terms of energy consumption (mJ).

The results of the CPU speed-based task allocation show that it consumes less energy than the random-based solution. The reason for less energy consumption is that the proposed solution enables the controller node in a way that always assigns the task to a high-speed mobile device that executes it quickly. Because of the fast execution time, energy consumption is also minimized. However, in random-based task allocation, the controller node allocates the task to the slower device as it does not consider the compute node resources that prolong its execution time and also consumes more energy. Thus, the CPU speed-based task allocation solution helps to save energy up to 69.78%.

The results of a core-based task allocation are also more promising than random-based task allocation in terms of energy consumption. In random-based task allocation, resources are not usually utilized in an efficient way, which wastes energy. However, core-based task allocation ensures efficient resource utilization by enabling the controller to select compute nodes that have more cores. Thus, the selection of such devices helps to execute the task in a quicker manner than random-based task allocation because of the former's high processing capabilities. In this way, the core-based solution helps to reduce energy consumption up to 68.25% compared with random-based task allocation.

The workload-based task allocation solution enables the controller node to select only the device that has a lighter workload running in the background. Based on the workload criteria, much more energy can be saved than random-based task allocation. In random-based task allocation, the larger tasks can be allocated to such devices that already have lots of workloads running in the background, which causes higher energy consumption. However, the workload-based selection of the compute node helps to minimize the energy consumption. Thus, the results show that workload-based task allocation helps to save energy up to 69.06%.

The results of the two parameters-based (CPU speed and workload) task allocation reveal that it helps to reduce energy consumption when compared with random-based task allocation. Although performing task allocation based on two parameters helps to reduce energy consumption up to 67.26% compared with random-based task allocation, single parameter-based task allocation helps to save more energy. The reason is that through single parameter-based task allocation, the tasks are being executed only on one device that usually has high specifications, whereas through multiple parameters-based task allocation, tasks are usually allocated to more than one device as noted in our simulation, where specifications of the compute nodes were quite diverse.

In that simulation scenario, although execution time could be minimized due to a time slots factor compared with single parameter-based task allocation, energy consumption could not be minimized in comparison with single parameter-based task allocation. Such is also the case with the two and three parameters-based task allocation solutions (SW and SCW). Although the results of SW and SCW show that the solutions save energy consumption up to 67.26% and 57.33%, respectively, these results were not better than two single parameter-based task allocation solutions, such as SO and WO. Hence, the discussion concludes that proposed heterogeneity-aware task allocation solutions use less energy than the random-based task allocation. In addition, the CPU speed-based task allocation is one of the most energy-efficient solutions in our simulation scenario.

IX. SCOPE AND LIMITATIONS

The proposed heterogeneity-aware task allocation algorithms are effective for all interactive MAC-based compute-intensive task execution. To minimize execution time and energy, the MAC controller node can adopt these solutions to make task allocation decisions. Moreover, cloudlet based applications can also take advantage of these algorithms in a particular case when execution is performed in a distributed manner.

Despite many advantages, there are some limitations of the proposed algorithms. For example, in single parameter-based proposed solutions (e.g., SO and CO) all the tasks

are allocated to only one compute node, which may not be feasible and can lead to wastage of resources. Moreover, the proposed solutions require complete device specification and workload information in real-time, which may pose significant overhead on the controller. Furthermore, one of the limitations is that once the controller collects all the information of the specifications and workload from the compute nodes, it makes task allocation decisions based on the defined policy in the proposed algorithms. However, the workload information may be changed in between the task assignment process on the compute nodes. Thus, in such a situation, the proposed heterogeneity-aware task allocation solutions do not incorporate any feature that enables the controller to change the task allocation policy at the run time.

X. CONCLUSION AND FUTURE DIRECTIONS

This paper has proposed five heterogeneity-aware task allocation solutions to address the issue of longer execution time and greater energy consumption by enabling the controller node to make an efficient task allocation decision. The proposed solutions have minimized the task execution time and energy consumption for MAC-based task execution. The proposed solutions were implemented using a controller node that is responsible for allocating the task in the MAC environment. The developed system model has been validated by comparing the empirical execution time results obtained from the five proposed solutions. The differences between the execution time results obtained from the system model and each proposed solution were not significant. Performance evaluation of the proposed solutions have also been done. The results revealed that heterogeneity-aware task allocation solutions, such as SO, CO, WO, SW, and SCW outperformed the random-based solution by reducing the execution time up to 56.72%, 53.12%, 56.97%, 61.23%, and 71.55%, respectively. In addition, these heterogeneity-aware task allocation solutions saved energy up to 69.78 %, 69.06 %, 68.25 %, 67.26 %, and 57.33 %, respectively. Based on the obtained results, we conclude that the proposed solutions help to improve task performance in terms of execution time and energy consumption, in comparison with random-based solution.

Node turnover is an important factor to consider during task allocation in MAC, as the mobile nodes may leave and join at any time. For example, a compute node with an allocated task may leave the MAC without reporting back the results. In such situations, task reallocation needs to be investigated. Moreover, new solutions need to be devised to engage newly joined compute nodes. Battery power is another important consideration during task assignment.

REFERENCES

[1] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Commun. Mobile Comput.*, vol. 13, no. 18, pp. 1587–1611, Dec. 2013.

[2] E. Ahmed, A. Gani, M. K. Khan, R. Buyya, and S. U. Khan, "Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges," *J. Netw. Comput. Appl.*, vol. 52, pp. 154–172, Jun. 2015.

[3] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, and S. Guizani, "Mobile ad hoc cloud: A survey," *Wireless Commun. Mobile Comput.*, vol. 16, no. 16, pp. 2572–2589, Nov. 2016. [Online]. Available: http://dx.doi.org/10.1002/wcm.2709

[4] M. Whaiduzzaman, A. Naveed, and A. Gani, "MobiCoRE: Mobile device based cloudlet resource enhancement for optimal task response," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2016.2564407.

[5] U. Shaukat, E. Ahmed, Z. Anwar, and F. Xia, "Cloudlet deployment in local wireless area networks," *J. Netw. Comput. Appl.*, vol. 62, pp. 18–40, Feb. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2015.11.009

[6] E. Ahmed, A. Akhunzada, M. Whaiduzzaman, A. Gani, S. H. Ab Hamid, and R. Buyya, "Network-centric performance analysis of runtime application migration in mobile cloud computing," *Simul. Model. Pract. Theory*, vol. 50, pp. 42–56, Jan. 2015.

[7] N. Fernando, S. W. Loke, and W. Rahayu, "Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2011, pp. 281–286.

[8] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proc. 1st ACM Workshop Mobile Cloud Comput. Services, Social Netw. Beyond*, 2010, p. 6.

[9] S. Ghasemi-Falavarjani, M. Nematbakhsh, and B. S. Ghahfarokhi, "Context-aware multi-objective resource allocation in mobile cloud," *Comput. Elect. Eng.*, vol. 44, pp. 218–240, May 2015.

[10] B. Li, Y. Pei, H. Wu, and B. Shen, "Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds," *J. Supercomput.*, vol. 71, no. 8, pp. 3009–3036, Aug. 2015.

[11] B. Zhou, A. V. Dastjerdi, R. Calheiros, S. Srirama, and R. Buyya, "mCloud: A context-aware offloading framework for heterogeneous mobile cloud," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2015.2511002.

[12] Z. Lu, J. Zhao, Y. Wu, and G. Cao, "Task allocation for mobile cloud computing in heterogeneous wireless networks," in *Proc. 24th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2015, pp. 1–9.

[13] T. Shi, M. Yang, X. Li, Q. Lei, and Y. Jiang, "An energy-efficient scheduling scheme for time-constrained tasks in local mobile clouds," *Pervas. Mobile Comput.*, vol. 27, pp. 90–105, Apr. 2016.

[14] M. Chen, Y. Hao, Y. Li, C.-F. Lai, and D. Wu, "On the computation offloading at ad hoc cloudlet: Architecture and service modes," *IEEE Commun. Mag.*, vol. 53, no. 6, pp. 18–24, Jun. 2015.

[15] E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using MapReduce," School Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Rep. CMU-CS-09-164, 2009.

[16] X. Guo, L. Liu, Z. Chang, and T. Ristaniemi, "Data offloading and task allocation for cloudlet-assisted ad hoc mobile clouds," *Wireless Netw.*, pp. 1–10, 2016, doi:10.1007/s11276-016-1322-z.

[17] M. Shiraz and A. Gani, "A lightweight active service migration framework for computational offloading in mobile cloud computing," *J. Supercomput.*, vol. 68, no. 2, pp. 978–995, May 2014.

IBRAR YAQOOB received the B.S. (Hons.) degree in information technology from the University of the Punjab, Gujranwala campus, Pakistan, in 2012. He is currently pursuing the Ph.D. degree in computer science with the University of Malaya, Malaysia, since 2013. He has authored number of research articles in refereed international journals. His numerous research articles are very famous and most downloaded in top journals. His research interests include big data, mobile cloud, Internet of things, cloud computing, and wireless networks. He received the scholarship for his Ph.D. degree and a Bright Spark Program Research Assistant.

EJAZ AHMED was a Research Associate with the Cognitive Radio Research Laboratory, SEECS, NUST, Pakistan, from 2009 to 2012, and with the Center of Research in Networks and Telecom, MAJU, Pakistan, from 2008 to 2009. He is currently a Senior Researcher with the Center for Mobile Cloud Computing Research, University of Malaya, Malaysia. His research experience spans over ten years. He has successfully published his research work in over thirty reputable international journals and conferences. His areas of interest include mobile cloud computing, mobile edge computing, Internet of things, and cognitive radio networks. He is an Associate Technical Editor of the *IEEE Communications Magazine*, the IEEE Access, and Springer MJCS. He is also serving as a Lead Guest Editor of the *Elsevier Future Generation Computer Systems Journal*, the IEEE Access, the *Elsevier Computers and Electrical Engineering*, the *IEEE Communications Magazine*, the *Elsevier Information Systems* and *Transactions on Emerging Telecommunications Technologies*.

ABDULLAH GANI (M'01–SM'00) received the bachelor's and master's degrees from the University of Hull, U.K., and the Ph.D. degree from the University of Sheffield, U.K. He is a currently Full Professor with the Department of Computer System and Technology, University of Malaya. He has vast teaching experience due to having worked in various educational institutions locally and abroad, including schools, teaching college, the Ministry of Education, and universities. His interest in research started in 1983 when he was chosen to attend a Scientific Research course in RECSAM by the Ministry of Education, Malaysia. He has authored over 150 academic papers in conferences and respectable journals. He actively supervises many students at all level of study-bachelor, master, and Ph.D. His interest in research includes self-organized systems, reinforcement learning, and wireless-related networks. He is currently involved in mobile cloud computing with a High Impact Research Grant of U.S. $1.5 million for the period of 2011–2016.

SALIMAH MOKHTAR is currently an Associate Professor with the Department of Information System, Faculty of Computer Science and Information Technology, University of Malaya in Malaysia. Her research interests are in the area of Information System for education, blended learning, scholarship of teaching and learning, spiritual intelligence, big data, data science, and learning analytics.

MUHAMMAD IMRAN was a Post-Doctoral Associate on joint research projects between KSU, University of Sydney, and Iowa State University, USA. He is currently an Assistant Professor with the College of Computer and Information Sciences, King Saud University (KSU), since 2011. He is also a Visiting Scientist with Iowa State University. His research interest includes mobile ad hoc and sensor networks, WBANs, M2M, IoT, SDN, and fault tolerant computing. He has authored over 85 publications in refereed international conferences and journals. His research is financially supported by several grants. The European Alliance for Innovation (EAI) has appointed him as a Co-Editor in Chief of the *EAI Transactions on Pervasive Health and Technology*. He also serves as an Associate Editor of the IEEE Access, the *Wireless Communication and Mobile Computing Journal* (SCIE, Wiley), the *Ad Hoc and Sensor Wireless Networks Journal* (SCIE), the *IET Wireless Sensor Systems*, the *International Journal of Autonomous and Adaptive Communication Systems* (Inderscience), and the *International Journal of Information Technology and Electrical Engineering*. He served/serving as a Guest Editor of the *IEEE Communications Magazine* (SCIE), the *Computer Networks* (SCIE, Elsevier), the *MDPI Sensors* (SCIE), the *International Journal of Distributed Sensor Networks* (SCIE, Hindawi), the *Journal of Internet Technology* (SCIE), and the *International Journal of Autonomous and Adaptive Communications Systems*. He has been involved in over 45 conferences and workshops in various capacities such as a chair, co-chair and technical program committee member. These include the IEEE ICC, the Globecom, the AINA, the LCN, the IWCMC, the IFIP WWIC, and the BWCCA. He has received number of awards such as the Asia Pacific Advanced Network fellowship.