Reliable resource allocation and fault tolerance in mobile cloud computing

Zahra Najafabadi Samani
Department of Computer Architecture, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran
najafabadizahra@gmail.com
Mohammad Reza Khayyambashi
Department of Computer Architecture, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran
M.R.Khayyambashi@eng.ui.ac.ir

Received: 08/Jul/2019 Revised: 11/Nov/2019 Accepted: 24/Dec/2019

Abstract
By switching the computational load from mobile devices to the cloud, Mobile Cloud Computing (MCC) allows mobile devices to offer a wider range of functionalities. There are several issues in using mobile devices as resource providers, including unstable wireless connections, limited energy capacity, and frequent location changes. Fault tolerance and reliable resource allocation are among the challenges encountered by mobile service providers in MCC. In this paper, a new reliable resource allocation and fault tolerance mechanism is proposed in order to apply a fully distributed resource allocation algorithm without exploiting any central component. The objective is to improve the reliability of mobile resources. The proposed approach involves two steps: (1) Predicting device status by gathering contextual information and applying TOPSIS to prevent faults caused by volatility of mobile devices, and (2) Adapting replication and checkpointing methods to fault tolerance. A context-aware reliable offloading middleware is developed to collect contextual information and manage the offloading process. To evaluate the proposed method, several experiments are run in a real environment. The results indicate improvements in success rates, completion time, and energy consumption for tasks with high computational loads.

Keywords: Mobile Cloud Computing; Fault tolerance; Reliability; Replication; Checkpointing.

1- Introduction

As a result of the recent developments in mobile technologies, mobile devices (e.g., smartphone and tablet PC) have become an integral part of human life as highly effective and convenient means of communication. However, mobile devices face an array of challenges in terms of both resources (e.g., battery life, storage, and bandwidth) and communications (e.g., mobility and security). To overcome this limitation, offloadding computations from mobile devices to cloud is proposed [1, 2]. A three-tier architecture is defined for Mobile Cloud Computing (MCC) including remote cloud servers, local servers known as cloudlets, and adjacent mobile devices. Offloading to remote servers can be costly and introduces latency. Besides, they are not always available and cloudlets limit mobility of mobile devices [1, 3-6]. To address this issue, in this paper, the third tier of the MCC (consist of neighboring mobile devices) are consider where both service requester and service providers are mobile devices.
However, there are several issues in mobile devices as resource providers such as unstable wireless connection, limited energy capacity and frequent changes of location. Thus, fault and fault tolerance are among the main challenges faced by mobile resource providers, which should not be ignored. Faults in the offloading process are primarily caused by energy constraints, mobility, and availability associated with mobile devices. Many previous works try to select appropriate resource providers among available mobile devices to allocate tasks by gathering contextual information [6-8]. However, they mostly fail to address fault prevention and tolerance. Others consider only prevention of fault [3, 13, 18-21] or fault tolerance [16, 14]. Moreover, limited energy has not been investigated as a factor of fault prevention [11-13] or only replication techniques are applied in fault tolerance [14-17]. It is well noted that replication techniques are very costly in high computational load. The objective here is to maximize the success rate of the offloading process so that some Quality of Service (QoS) constraints are satisfied. In this paper, a fully-distributed reliable resource allocation algorithm is used wherein, energy, mobility and availability of mobile devices are considered as fault factors. This approach promotes system robustness by predicting the states of mobile devices to avert the faults caused by mobile volatility. Then depending on the size of the task, checkpointing or replication methods are used as the fault tolerance methods. In order to provide dynamic and accurate resource allocation and to predict the states of mobile devices, contextual information needs to be

gathered from devices, applications, and the environment to be applied in decision-making. A context-aware offloading middleware is developed to collect contextual information and manage the offloading process. To evaluate this new proposed method, our experiments are run in a real environment.

The rest of the paper is organized as follows: Section 1 reviews the related works. In Section 2, a reliable system is formulated and solved using a two-stage approach. Section 3 pertains to ranking and classification of mobile devices using TOPSIS. In Section 4, the architecture of the context-aware reliable offloading middleware is described. Section 5 presents the evaluation results of the proposed approach. Finally, the paper is concluded in Section 6.

2- Related work

There are many studies that investigate the task allocation problem by collecting contextual information in MCC [6-8]. However, those do not consider fault and fault-tolerant methods. Shi et al. [7] propose Serendipity which enables offloading through gathered information profile and releasing two versions of task allocation algorithm, energy aware Serendipity and time optimizing. [6] considers the local tier of mobile cloud where both service requester and service providers are mobile. The paper proposes a resource allocation algorithm with multi-objective optimization to minimize completion time and energy consumption of all participating mobile devices. Ref. [8] considers all three layers of MCC and proposes a context-aware offloading decision algorithm to derive an optimal offloading decision under the context of the mobile device and cloud resources.

Several studies consider fault in MCC. However, some of them consider only prevention of fault and none of fault tolerance method (e.g. replication or checkpointing) is adopted after failure occurrence [3, 12, 13, 18-21].

In [18, 19], a monitoring approach based on Markov chain are proposed for analyzing and predicting states of resources. They propose a monitoring time interval rate in order to monitor the correct state information of mobile resources. In these papers, the manner of applying monitoring information in fault is missed. In the context of mobile cloud, Ref. [20] considers mobile devices as resources and improves mobile resource efficiency through energy-aware management by selecting appropriate mobile devices. In this scheme, firstly devices are divided into four groups in terms of efficiency and mobility based on a threshold value. The devices in each group are then ranked. To prevent faults, no tasks are assigned to unreliable mobile devices. In [13, 21], a dynamic grouping scheme is presented to manage mobile devices in MCC. In [13] availability and mobility, and in [21] mobility and efficiency are considered as fault factors. Then, cut-off points are adopted by entropy. Next, according to the cut-off points, mobile devices are arranged into several groups. In these works, mobile devices' energy is not considered as fault factor in grouping. In all these papers [13, 18-21], a central component is applied to monitor and manage mobile devices that cause a single failure and limit the mobility of mobile devices.

Ref. [12] proposes a reliable resource allocation method according to availability and mobility of mobile devices in MCC. Initially, subtasks are assigned to mobile devices with minimal mobility and then resources with the highest availability are selected. Despite being a limitation, energy is not considered as a fault factor. In [6], a context-aware offloading scheme for mobile peer-to-peer environments is proposed in which both servers and clients are mobile. Their scheme chooses adjacent reliable mobile devices to assign subtasks in order to prevent fault. This dissertation just supports fault occurrence by mobility and ending energy of mobile.

On the other hand, in [14-17] fault tolerance is achieved using only replication which is generally not very efficient for task with high computational loads, imposes high costs, and occupies many resources. In addition, in [16, 14], to prevent fault occurrence, the system does not select adjacent reliable mobile devices to assign subtasks. In Hyrax [16], a Hadoop-based platform is proposed that supports cloud computing on smartphones. Replication is used for fault tolerance; subsequent to failure, failed subtasks are re-executed without user intervention. Although Hyrax provides high scalability, the system exhibits poor performance with CPU-intensive tasks. The central server in Hyrax causes a bottleneck in the system and limits the mobility of mobile devices. Ref.[14] presents the implementation of a platform for providing fault tolerance in MCC. Their approach improves reliability by the use of a dynamic and adaptive replication which uses the minimum number of replicas. In this model, a new replica is placed on the most reliable node until the required reliability level is reached.

Ref. [15] considers fault tolerance and quality of service in social mobile cloud computing environment by using Content Addressable Network. In this paper the cloud server selects the best resource from the adjacent mobile devices in terms of quality of public and mobile device services and, replication is used for fault tolerance. Nevertheless, energy and mobility are not considered as a fault factor in the paper. In [17], a framework is proposed to support fault tolerance which integrates the k-out-of-n reliability mechanism into mobile cloud formed only by mobile devices. The framework provides services for applications that aim to reliably store and process data in the mobile cloud such that the energy consumption for retrieving and processing the data is minimized. An

unrealistic assumption in this work is that the nodes have equal energy consumption for processing. It is possible that subtasks are allocated to nodes with high-energy consumption, so these nodes will fail in the future.
[9] applied the fault tolerance technique to handle the faulty VMs in the MCC using the human disease resistance mechanism which identifies the faulty virtual machines and reschedules the tasks to the identified suitable virtual machines. In this paper, they do not consider fail in mobile devices as resources.

In [11, 10], a dynamic classification scheme for reliable management of mobile devices as resources in MCC is used. In [11] mobile devices are grouped according to availability and mobility by adopting entropy, and in [10] mobile devices are classified into groups based on their processing capacity, availability, and communication condition by adopting tree learning. Then, checkpointing and replication are applied to groups with high and low reliability, respectively. However, in these works, administrative tasks are handled by a proxy which is in conflict with nature of mobile networks and mobility of mobile devices. Moreover, task assignment has not been determined. In addition, in [11] energy is not considered as a fault factor, and [10] considers only the remaining battery of the devices and study of mobility model and trajectory is overlooked.

3- Proposed Reliable System Model

Reliability is one of the most important aspects in mobile cloud computing due to mobility and resource constraints of mobile devices. Faults and failures should be managed and controlled in an active manner to minimize the effects of failures on the system. There are two complementary approaches to establish reliability: (1) Fault Prevention and (2) Fault Tolerance [23].
In the proposed model, each task has n independent subtasks $\{T= t_i \ | 1\leq i\leq n\}$. And the mobile cloud environment is formed by h mobile device $\{S=s_k | 1\leq k\leq h\}$(where s_h is a client mobile device that does task offloading). In this model, at any given time, only one offloading is executed, applications are segmented before, and offloadable subtasks are independent and executed in parallel.

3.1- Fault Prevention

The goal of fault prevention is to prevent system failure by ensuring that all possible causes of unreliability are removed [23]. One of the challenges in mobile cloud is selecting reliable resources from adjacent mobile devices to assign tasks to and prevent system failure. The energy, mobility and availability factors of mobile devices result in more frequent system faults. Due to dynamic changes which disrupt mobile device applications, these features increase device load and decrease system performance. To the best of our knowledge, these factors have not been jointly addressed. In this paper, we seek to prevent system failure by selecting reliable devices according to three important factors which cause system faults: energy, mobility and availability.

3.1.1- Energy

Battery energy is among the main constraints in mobile devices. As battery life is inherently limited, it may run out at any moment, causing premature job termination. Hence, mobile devices must preserve battery power during and after processing a job. In mobile cloud environments, subtasks should be distributed in a manner that network lifetime is maximized and fault caused by battery energy limitations are avoided. This proposed energy model is inspired by [6]. In This model energy consumption and remaining energy levels of all nodes involved in offloading are considered. Variable notations are shown in Table 1.

Table 1. Definitions of notations.

n	Number of subtasks
h	Number of mobile devices
R_j	jth resource provider
t_j	Time needed to execute subtasks on Rj
e_j	Energy consumption per second running each subtask on Rj
et_j	Energy consumption on Rj to transfer one unit of data
$E0_j$	Initial energy level of Rj
Vin	Input size of subtask
$Vout$	Output size of subtask
b_j	Number of subtasks on Rj
E_j	Energy consumption on Rj

The energy consumptions of both the server and the client are indicated through Equ. (1). In the client side ($j = h$), energy consumption includes energy consumption of running local subtasks, transferring offloaded subtasks to nearby mobile devices, running the resource allocation algorithm. The energy consumption in server sides consists of resource provider Rj ($1\leq j<h$) consists of energy consumed for running the assigned subtasks and transmitting the results.

$$predicted\ E_j = \begin{cases} b_j * (t_j * e_j) + \sum_{j=1}^{h-1} b_j * (et_j * vin) + algE, & j = h \\ bj * [(t_j * e_j) + (et_j * vout)], & 1 \leq j < h \end{cases} \quad (1)$$

Then, Equ. (2) is applied to avoid allocating subtasks to nodes with low remaining energy. Here, the network's lifetime increases and the risk of energy depletion during processing is prevented.

$$E0_j - predicted\ E_j \geq \alpha \quad for\ all\ j= 1, \ldots, h-1 \qquad (2)$$

Next, Equ. (3). allows the subtasks to be allocated to the nodes according to their energy level:

$$Energy_j = \frac{predicted\ E_j}{E0_j - predicted\ E_j} \qquad (3)$$

In this model, in addition to taking care of nodes with low energy levels, the subtasks are fairly distributed among other nodes. We aim to maximize the lifetimes of nodes having the low remaining energy and high energy consumption rates. Here, network energy is reduced and no task is assigned to nodes with low remaining energy, which prevents failures caused by battery drain and increases minimal remaining energy of nodes. In [17], only remaining energy and transferred energy among nodes is considered while energy requirements are assumed to be equal for all nodes, which is not a realistic assumption.

3.1.2- Mobility Factor

Mobile devices as resource providers can join and leave the Mobile Cloud environment in an unpredictable manner. This interrupts the operation and may cause a system failure. Given their low reliability, it is difficult to consider mobile devices as resources. In this study, prediction colocation between client and server mobile devices is used to avoid the failure by mobility during offloading. In fact, a colocation time between two users is the time that two users visit each other and stay together. Many studies use a temporal mean for predicting colocation time [13, 21, and 22]. In this technique, time is split in slots, and a temporally varying mean of the encounters with other user is kept. Under this assumption, the estimated colocation time is accurate since user paths are well defined. However, it may present issues when applied to new environments. To overcome this issue, in this paper, the spatial-temporal mean is adopted to predict colocation time between server and client that follows the same principle, but taking into consideration the activity and place when the encounter between users take place. The anticipated colocation and movements of users are based on intermittent user behaviors to see a place and move among locations. Users periodically see a specific location and regularly stay there, allowing their routes and stationary time intervals to be estimated [3, 13, 22, and 24]. Here, a sequence of places *p* with time stamps *ten* and *tex* is stored for each user. The tuple $<p_i, ten_i, tex_i>$ indicates that user *u* enters place p_i (*p* is the place covered by one cellular tower or AP) at time ten_i and exits at time tex_i. A trajectory information group G_k is generated which contains *k* tuples. Each time the user moves from a location to another, the trajectory information is updated. This sequence is expressed by Equ. (4).

$$G_k = \{<p, ten, tex> | <p_{i-k}, ten_{i-k}, tex_{i-k}>, \ldots, <p_{i-1}, ten_{i-1}, tex_{i-1}>, <p_i, ten_i, tex_i>\} \qquad (4)$$

In the proposed scheme, first, client device movements are predicted. Markov chain is perhaps the most widely used model for human mobility due to its simplicity and efficiency. Many recent works adopt the approach for location prediction [25-28]. A Markov chain is a sequence of random variables X_t, which represent the states of a system, provided that the current state (X_t) depends only on the previous state (X_{t-1}) [29]. This could be expressed through Equ. (5).

$$P(X_t = x | X_0 = x_0, X_1 = x_1, \ldots, X_{t-1} = x_{t-1}) = P(X_t = x | X_{t-1} = x_{t-1}) \qquad (5)$$

The chain could also be depicted using a directed graph, whose edges are labeled with the probability of going from one state to another, from time *t - 1* to time *t*, Fig. (1).

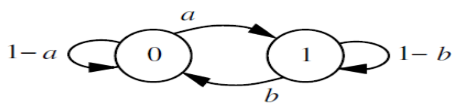


Fig 1.Markov chain model

In this study, the probability of going from location *i* to location *j* is calculated according to [28]. Here, locations are considered as states of the Markov chain. As mentioned earlier, in a Markov chain, the states are independent; in other words, the location subsequent to *i* is not dependent on the history of visited locations. Accordingly, at processing time, the probability of a transition from *i* to *j* equals the number transitions from *i* to j at processing time is divided by the total number of outgoing transitions from *i* (i.e. the definition of probability). Respective probabilities are calculated on the client mobile device (during processing) by applying Equ. (6).

$$P(l_{t+1} = p_j | l_t = p_i) = \frac{n(l_{t+1} = p_j | l_t = p_i)}{\sum_{d=1, d\neq i}^{k} n(l_{t+1} = p_d | l_t = p_i)} \qquad (6)$$

The location with maximum probability is selected as the next location of the client during the process. The neighbor nodes are then informed of the client's next location. Adjacent nodes that are willing to cooperate, calculate their own colocation probabilities with the client node. Equ. (7) is applied to calculate the colocation between the service provider and the client after it is determined that the client node is going to stay in its current location during the process.

$$\mathrm{P}(l_{t+1} = p_i | l_t = p_i, tex_i - \mathrm{tc} \geq \mathrm{tre}) = \frac{\mathrm{n}(l_{t+1} = p_i | l_t = p_i, tex_i - \mathrm{tc} > \mathrm{tre})}{\sum_{d=1,d\neq i}^{k} n(l_{t+1} = p_d | l_t = p_i)} \quad (7)$$

In this equation, according to the characteristic of Markov chain and the definition of probability, the number of times the service provider visits the current location at processing time for *tre* is divided by total number of times it moves from i to any other location at processing time. Here, *tre* is the time required for processing. The variable *tc* denotes current time. The non-equation $tex_i - tc > tre$ indicates that service providers need to remain in the vicinity of the client as long as the task continues to execute.

Once it is determined that the client node intends to move to another location during the process, Equ. (8) Calculates the colocation between the service provider and the client. To find the colocation probability, the number of times the service provider has moved from i to j (along with the client during the process) is divided by the number of its movements from i to any other location. The non-equation $(tex_i - \mathrm{tc}) + (tex_j - ten_j) \geq \mathrm{tre}$ means that the colocation time must exceed the time necessary for processing.

$$\mathrm{P}\left(l_{t+1} = p_j \middle| l_t = p_i, (tex_i - \mathrm{tc}) + (tex_j - ten_j) \geq \mathrm{tre}\right) = \frac{\mathrm{n}\left(l_{t+1} = p_j \middle| l_t = p_i, (tex_i - \mathrm{tc}) + \left(tex_j - ten_j\right)\right.}{\sum_{d=1,d\neq i}^{k} n(l_{t+1} = p_d | l_t = p_i)} \quad (8)$$

Finally, service providers send their corresponding colocation probabilities to the client node. In this manner, the best resources from the adjacent mobile devices are automatically chosen. The best resources have the most chance of staying in colocation.

3.1.3- Availability

Availability can have several meaning according to the system requirements. In the context of mobile devices used as resources, it refers to the probability that a mobile device performs and replies correctly while acting as a resource. High availability systems aim to minimize downtime and repair costs. In mobile applications, availability is a crucial requirements and an achievable objective. In the context of MCC or mobile grid, mobile devices are often classified according to availability [11-13]. In [11], the number of faults caused by a mobile device is used for availability information. Accordingly, in this paper the success rate is applied to obtain availability. The higher values in the past predict greater availability and success in the future. Each service provider calculates its availability using Equ. (9). and the resulting values are transmitted to the client mobile device.

$$Availability = \frac{number\ of\ successful\ tasks}{\text{total number of tasks}} \quad (9)$$

The client then selects the devices with the highest success rate (i.e. availability) as the resources. Thus, tasks are assigned to devices that are more reliable and devices with high failure rates are avoided.

3.1.4- Estimating context information

There are several techniques to predict runtime, energy consumption, and output size of a subtask on a mobile device [32, 33]. We use a dual profiling approach inspired by [33], which consists of a peer-centric and a task-centric profile. The former is a history-based profile maintaining the runtimes and energy consumptions of the last *n* runs of a subtask on a specific peer. Here, the average profile data from the last ten runs is calculated as an estimate. Since the mobile cloud is a dynamic environment and there may be new to which the task has never been assigned, a task centric profile is used for new devices. In this approach, a device is selected as the base; then, by comparing the processing power of the base device with that of the new device, subtask energy consumption and runtime on the new device are estimated.

3.2- Fault Tolerance

A fault-tolerant system should be able to manage defects in hardware or software components as well as other unexpected downtimes. Despite fault prevention approaches, failure is inevitable; therefore, it is necessary to take appropriate measures after system fault [23]. There are numerous methods for achieving fault tolerance in distributed systems, among which replication and checkpointing are most popular.

3.2.1- Active Replication

Replication techniques replicate similar tasks, which can be run in a simultaneous manner on several devices. When one of the replicas fails another performs its task [30]. In this paper, active replication is used to promote resources reliability against faults caused by the volatility of mobile devices for tasks with low computational load. The reasons for this choice are: (1) active replication is a non-centralized technique which suits the fully distributed design of the proposed system; (2) Failures are fully hidden from the clients, since requests are still processed even if one replica fails; (3) short response time even in case of failures because each replica works independently which is important for mobile users; and (4) simplicity which is important for mobile devices with limited resources.

In the proposed method once a service provider fails for any reason, another replica is responsible for the client node requests. When the client node receives the first response from a replica, others halt processing in order to reduce energy consumption. Although replication techniques have many advantages with high fault tolerance, they are costly for tasks with high computational loads.

3.2.2- Independent Checkpointing

Once a failure occurs, it is vital to restore the process to its correct state. So, in this paper, independent checkpointing is applied for tasks with high computational load. This is better for independent processes and reduces the computational overhead [31]. Independent checkpointing mainly involves restoring the system from its present erroneous state back into a previously correct state. This requires the state of the system to be occasionally recorded so that, when faults occur, the state can be restored. The state of the system is stored on the client mobile devices at regular intervals. When a subtask fails, that subtask along with the previously correct state, is assigned to another reliable service provider without user intervention.

3.3- Reliable task allocation in the Mobile Cloud

In the mobile cloud, when a client mobile device wants to run a compact application, it first asks for an offloading service. Next, it applies service discovery to identify adjacent mobile devices by transmitting a broadcast message containing its location during the service. As the service provider receives the offloading request message, it calculates the energy consumption ratio in relation to residual energy after performing the subtask through Equ. (3). Then, it calculates its colocation probability with the client and availability using Equs. (7-8) and Equ. (9), respectively. Finally, it sends this information along with its energy consumption rate and current energy in response to the client. Algorithm 1 outlines the execution algorithm of the subtasks on the service provider.

Algorithm 1: Execution Subtasks Algorithm

```
Numet=0;//number of all execution tasks
Numset=0;//number of all successfully executed tasks
Collect location information;
Receive request from client;
If mobile device is willing to cooperate then
   Collect energy information;
   Calculate context information();
   Send context-inf to client;
   Receive subtasks;
   If subtask is large then
      While(Receive request for checkpoint)
         Send checkpoint;
      If chechpoint ≠ 0 then
         Execute subtask from checkpoint;
         Send result;
   Else
     Execute subtask;
     Send result;
   Receive ack from client;
   If ack==1 then
      Numet++;
      Numset++;
   Else
      Numet++;
Function Calculate context information()
   Calculate colocation probability with client(p-location);
   Calculate consumable battery(cb);
   Calculate ratio of consumable battery to remaining battery(E);
   Calculate availability;
   Context-inf set(p-location, E, current E,cb, availability);
End
```

The client discovers its adjacent mobile devices and obtains their context information through messages sent by adjacent mobile devices in response to the client. First, the service providers' remaining energy levels are checked. If the levels are lower than the threshold from Equ. (1), the client does not assign any subtasks to these service providers. Next, the client predicts the providers' behaviors by using their context information. They are ranked accordingly by applying Equ. (10-18). The service providers are then partitioned into two groups of high and low reliability. Finally, in order to take advantage of fault tolerance techniques, according to the computational load of the task, either replication or checkpointing is adopted. For tasks with low computational load, active replication is employed and the subtasks in several groups with the highest reliability are replicated in a simultaneous manner.
Subtasks are allocated to groups according to the rank of devices through Equ. (13). More subtasks are assigned to devices with higher ranking. Every service provider executes the assigned subtasks and replies the results to the client. Once a subtask execution is finished on one of the replicas and the result is transmitted to the client, the subtask is no longer executed on other replicas. This contributes to saving resources and reducing traffic on the network. Replication in several devices for tasks with low computational load is not costly and occupies few resources; however, in such cases, restarting a subtask and taking checkpoints are of high overhead and cost. The decision-making algorithm on the client side is presented in Algorithm 2.

Algorithm 2: Dynamic Reliable Context-aware Decision making Algorithm

```
Function main()
   Collect location information; //call monitoring
   Request offloading();
   Decision offloading();
   Receive result;
   While  not receive result of all subtask do
   Request offloading;
   Receive result;
   Merge all results;
End
Function Request offloading()
   Calculate maximum job execution time ;
   Calculate next location in maximum job execution time ;
   Send broadcast with next location;
   While (true) do
      Receive reply from vicinal mobile devices;
      Context $\leftarrow$ Set context(energy, location,  availability, id);
   For i=1 to all vicinal device do
     Rank ⟵ TOPSIS(context);
     If Rank ≥ 0.5 then
       Group₁ ⟵ mobile device;
     Else
       Group₂ ⟵ mobile device;
   In Group₁
      Sort in ascending order of Rank;
   In Group₂
      Sort in ascending order of Rank;
End
Function Decision offloading()
    If subtasks is small then
      Call Replication;
    Else if subtasks is large then
      Call check pointing;
End
```

Replication is very costly for tasks with high computational load and occupies many resources. Consequently, in this paper, checkpointing is applied for tasks with high computational load. In checkpointing, first, subtasks are assigned to the group with the highest reliability according to device rankings. Next, at regular intervals, system state is stored on the client mobile device. Given the dynamic nature of the mobile cloud environment, after a device fails, client mobile device discovers its adjacent mobile devices again and receives their context information. Then once more, the client ranks and classifies adjacent mobile devices where the failed subtask with previously correct state is assigned to another reliable service provider without user intervention. Due to the high computational load of the subtask, the system replaces an erroneous state with an error-free state. Thus, the new service provider does not need to start from the beginning to process the failed subtask. This proposed approach saves time and cost of resources if a fault occurs in the final moments of processing.

Once the client receives a result from a service provider, it replies with an acknowledgment message to the sender, who then increments its number of successful tasks. Contrarily, if a subtask fails, it is re-executed on other adjacent mobile devices without user intervention. Ultimately, the client collects and merges all the results.

4- Ranking and grouping mobile devices with TOPSIS

In this study, TOPSIS is applied to rank and group mobile devices [34]. There are two reasons for this choice: (1) the concept of TOPSIS is rational, and (2) its algorithm is simple and light which is suitable for mobile devices with limited resources. In addition, TOPSIS can take objective weights into consideration in the comparison process. The technique includes a number of options and attributes for decision making. The options must be ranked according to these attributes. This solution procedure takes the following seven steps:

- **Step1.** Preparing the decision-making matrix with p rows and three columns where r_{pq} are the elements of this matrix, Equ. (10). The rows represent mobile devices in the vicinity of the client device while the columns represent the decision making attributes. Three attributes, namely energy consumption to the remaining energy, colocation probability between the service provider and the client and availability, are considered for ranking mobile devices.

$$D = \begin{array}{c} \\ \\ \\ \\ \end{array}\begin{array}{cccc|c} X_1 & X_2 & \cdots & X_q & \\ r_{11} & r_{12} & \cdots & r_{1q} & A_1 \\ r_{21} & r_{22} & \cdots & r_{2q} & A_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{p1} & r_{p2} & \cdots & r_{pq} & A_p \end{array} \tag{10}$$

- **Step2.** In this step, the decision-making matrix (D) is normalized by using Equ. (11).

$$n_{ij} = \frac{r_{ij}}{\sqrt{\sum r_{ij}^2}} \qquad i = 1,2,\dots,p \ , \quad j = 1,2,\dots,q \tag{11}$$

- **Step3.** Each attribute is assigned a weight based on the concept of Shannon entropy. Entropy is a major concept in the information theory and represents the amount of unreliability in a discrete probability distribution (random variable) [35]. Lower weights are assigned to attributes with identical values. This is because such attributes do not contribute to distinguishing options. They are, thus, less prominent. For each attribute, entropy is defined through Equ. (12).

$$E_j = -\frac{1}{\ln r}\sum_{i=1}^{p} n_{ij}\, ln\, n_{ij} \tag{12}$$

The degree of divergence dj (the contrast intensity of each attribute) and the weight for each attribute is indicated through Equ. (14) and (13), respectively.

$$d_j = 1 - E_J \tag{13}$$

$$W_j = \frac{d_j}{\sum_{k=1}^{q} d_k} \tag{14}$$

- **Step4.** This step involves the calculation of the weighted normalized decision matrix using Equ. (15).

$$v_{ij} = n_{ij} * w_j \quad i = 1,2,\dots,p \ , \quad j = 1,2,\dots,q \tag{15}$$

- **Step5.** In this step to rank alternatives, the solutions are compared with the positive ideal solution (A^+) and the negative ideal solution (A⁻), Equ. (16) (J and J' are the index sets of the benefit and cost attributes, respectively)

$$A^+ = \{(\min v_{ij} | j \in J), (max v_{ij} | j \in J') \,|\, i = 1,2,\dots,p\} = \{v_1^+, v_2^+, \dots, v_q^+\}$$
$$A^- = \{(\text{man}\, v_{ij} | j \in J), (min v_{ij} | j \in J') \,|\, i = 1,2,\dots,p\} = \{v_1^-, v_2^-, \dots, v_q^-\} \qquad (16)$$

- **Step6.** The distances of each solution from the ideal solution (d_i^+) and negative ideal solution (d_i^-) is calculated through Equ. (17).

$$d_i^+ = \sum_{j=1}^{3} (v_{ij} - v_j^+)^{\frac{1}{2}} \qquad i = 1,2,\cdots,p$$
$$d_i^- = \sum_{j=1}^{3} (v_{ij} - v_j^-)^{\frac{1}{2}} \qquad i = 1,2,\cdots,p \qquad (17)$$

- **Step7.** This step involves calculating ranks i.e. relative closeness of a solution to the ideal solution. The devices are ranked according to Equ. (18).

$$CL_i = \frac{d_i^-}{d_i^- + d_i^+} \qquad i = 1,2,\cdots,p \qquad (18)$$

Once mobile devices are ranked according to the aforementioned criteria, each device is placed in a group of high or low reliability based on its rank. Given the dynamic nature of mobile devices, their context information is continuously updated. Thus, devices are dynamically ranked and grouped. The number of subtasks assigned to each mobile device is determined according to its rank and reliability, Equ. (19).

$$nT_{R_i} = n \times \frac{CL_i}{\sum_{j=1}^{m} CL_i} \qquad i = 1,\cdots,m \qquad (19)$$

5- Context-aware reliable offloading middleware

In this section, a middleware is designed and implemented to employ reliable offloading and apply fault tolerance methods in the mobile cloud. This middleware collects contextual information, manages offloading steps, and adopts fault tolerance methods.

5.1- Architecture

This middleware consists of a client-side, which requests the offloading service, and a server-side, which provides services. A mobile device runs both parts, Fig. (2).

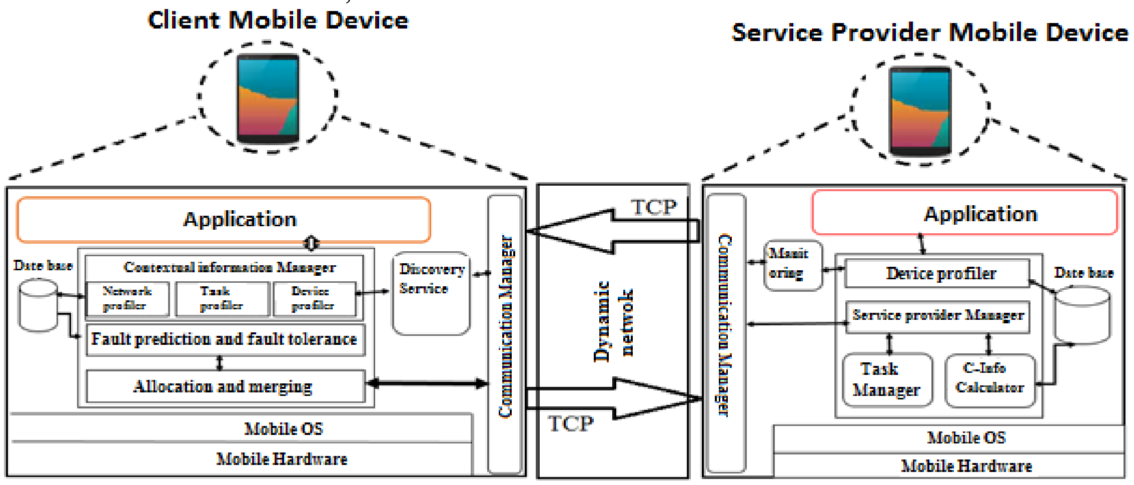


Fig 2. Context -aware reliable offloading middleware

5.1.1- Client-Side Middleware

The client-side middleware includes a discovery service, a contextual information manager, fault manager, Allocation and Merging, and a communications manager.

• Discovery Service: Adjacent mobile devices can be discovered using either pull or push techniques [18]. The former has a relatively small monitoring overhead, because the server's resource information is only requested when it is needed. Consequently, given resource constraints in mobile devices, and to reduce monitoring overhead, we propose a monitoring scheme which relies on the pull model. In this technique, when a client mobile device decides to offload a task to adjacent mobile devices, it sends a broadcast message, to which willing devices respond by transmitting their contextual information. This allows the client to discover its adjacent mobile devices and obtain their contextual information.

- Contextual information manager: This component gathers contextual information pertaining to tasks, resource providers, and the network through their aggregators i.e. task profiler, device profiler and network profiler, respectively.

a) The device profile contains energy consumption as well as remaining battery energy, trajectory information, total number of tasks assigned to the device, and the number of successfully performed by the device.

b) Task profile contains information about runtime and input/output size of each subtask. In this paper, it is assumed that application developers present offloadable parts of the task in the task profile.

- Fault manager: Initially, this component detects adjacent unreliable mobile devices; next, adjacent mobile devices are ranked according to their reliability and partitioned into groups of high or low reliability. Finally, in order to take advantage of fault tolerance techniques due to computational load of the task, either replication or checkpointing approach is applied.
- Allocation and Merging: this component assigns subtasks to adjacent mobile devices according to the fault manager component. In addition, this component merges the results. If a subtask fails or this component does not receive any result, the failure is reported to the fault manager component.
- Communications Manager: The Communications manager on the client provides network communication among client and service provider devices while monitoring this communication. In case the connection is disconnected, this component notifies the client to take appropriate fault tolerance measures depending on the condition.

5.1.2- Server-Side Middleware

This part of middleware includes a device profiler, a task manager, a context information calculator, a service provider manager, and a communications manager.

- Device profiler: Contextual information from the devices is collected and stored in a database on the device.
- Context information calculator: The three relevant criteria (i.e. colocation probability, energy ratio, and availability) are calculated based on the information received from the client and sent to the service provider manager component.
- Task manager: The component handles the request, executes the offloaded code, and relays the results to the service provider manager component.
- Service provider manager: It is responsible for coordinating the components on the service provider and following up processing on the server. Finally, this component collects the results.
- Communications Manager: The component handles communication between client and server on the server side, receives data and control data from the client, and sends context information and result to client.

6- Evaluation

In this section, the performance of the proposed system is evaluated by conducting real experiments on multiple mobile devices. A middleware is implemented into a library on Android operating system, which can be added in Android application for development. A face detection application is applied as the case study; it analyzes an assortment of photos as subtasks to identify the comprising faces. The application is implemented for the Android platform by applying android. media [36]. The middleware consists of approximately 6500 lines of Java code, which is used in following extensive experiments. In addition to the networking components, collecting and receiving checkpoints, collecting contextual information and tasks execution on this versions are run in separate threads. To evaluate the performance of the proposed method, a testbed of mobile devices is set up as show in Table 2. The mobile devices, which is creating a mobile cloud, are connected to an ad-hoc network using Wi-Fi with a mean bandwidth of 1.86 MBps. For each device, energy levels required for executing various tasks, the location of the device at any time, and availability are stored on its database. To measure the energy consumption of smartphones, we take advantage of PowerTour [37]. Furthermore, real-time device coordinates are obtained via GPS.

To demonstrate the performance of our proposed method, the proposed algorithm is compared with four other algorithms with different numbers of subtasks: random allocation, reliable allocation, replication, checkpointing algorithms, and dynamic grouping in [11]. In random allocation, mobile devices are randomly selected as resources without any measure of fault tolerance. Reliable allocation just aims to choose reliable mobile devices as resources and to assign tasks with no fault tolerance. The difference in the checkpointing algorithm lies in the application of checkpointing for fault tolerance. Another option is to use replication for fault tolerance, like [16, 14]. In these experiments, four criteria are applied to evaluate the performance of the proposed method:

Completion time: the amount of time to complete offloading plus the time of task allocation algorithm.

Success rate: indicative of successful offloading.

Consumed Energy: the total energy consumption of all devices involved in offloading. It is the sum of the values calculated for each node through Equ. (1).

Percentage of task failure: the percentage of subtasks that fail after being offloaded to adjacent mobile devices.

The impact of computational load (required processing time) of subtasks in this set of experiments is explored, where two sets of subtasks with different computational loads are of concern:
Case 1: Set of subtasks with low computational load.
Case 2: Set of subtasks with high computational load.

To evaluate the proposed algorithm, the two cases are examined in two scenarios, where two failure models are of concern [17]:
Fail-fast: a node fails at the first time-slot and cannot complete any task.
Fail-slow: a node may fail at any time; thus being able to complete some of its assigned tasks before the failure

Table 2. Features of mobile devices in testbed.

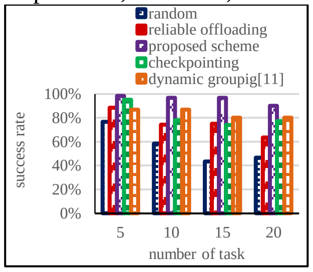
ID	***Mobile Device***	***CPU***	***Memory***	***OS***	***Battery capacity (Joule)***
A	Samsung Galaxy core 18260	Dual-Core 1.2GHZ Cortex-A5	1GB	Android OS, V4.1.2	24624 Joule
B	Samsung Galaxy Grand2	Quand-Core 1.2GHZ cortex-A7	1.5GB	Android OS, V4.4.2	35568 Joule
C	Samsung Galaxy Note 800	Quand-Core 1.4GHZ cortex-A7	2GB	Android OS, V4.1.2	100000 Joule
D	LG L Fino	Quand-Core 1.2GHZ cortex-A7	1GB	Android OS, V4.4.2	25992 Joule
E	Huawei Ascend G730	Quand-Core 1.3GHZ cortex-A7	1GB	Android OS, V4.4.2	31464 Joule

6.1- First Scenario

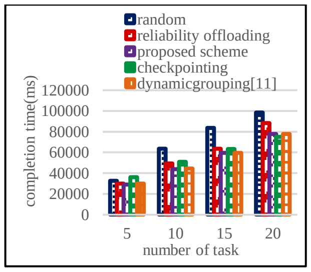
Here, the proposed method is evaluated and compared by considering fail-fast in both cases.
First Sample: The foregoing algorithms are reviewed and compared by considering fail-fast in case 1. In this situation, replication is applied in the proposed algorithm because the subtasks have low computational load.
Fig. (3.a) depicts the corresponding success rates. As evident, in all of the algorithms, the rate begins to suffer as the number of subtasks grows. This is because greater network traffic leads to higher failure rates. Likewise, an increase in the number of subtasks reduces the colocation probability between service provider and client node, thus increasing failure due to mobility. The random allocation algorithm has the lowest success rate while the proposed algorithm has the highest success rate with an average of 95.5%, which is 70%, 27%, 21%, and 14% higher than random, reliable offloading, checkpointing, and dynamic grouping, respectively. The reason is that the proposed algorithm selects reliable mobile devices with the highest rank as resources followed by replicating subtasks to several devices. However, the reliable allocation and checkpoint algorithms merely select reliable mobile devices without replicating subtasks in several mobile devices.
Completion times of the algorithms for the first sample can be seen in Fig. (3.b). The completion time in random allocation algorithm is higher than that of the other algorithms because it has a high failure rate forcing the failed subtasks to be reassigned. Contrarily, the proposed algorithm exhibits the lowest completion time, which is 19%, 6.5%, and 0.6% lower than random, checkpointing, and dynamic grouping, respectively. This is attributed to two reasons: (1) maximum success rate minimize the need to re-assign failed subtasks and (2) the response from the fastest replica is regarded as the ultimate result. The client may receive their results earlier than when the subtasks are not replicated. If the size of the task grows, checkpointing experiences a boost in performance because the overhead tends to dwindle. Applying checkpointing for big task reduces completion time. While for small task, overhead for getting checkpoint is large compared to the task size.
Total energy consumptions in the first sample are shown in Fig. (3.c). The proposed algorithm has the highest energy consumption, since it replicates tasks in several mobile devices. This increasing energy consumption is not significant because subtasks are small. Contrarily, reliable allocation and checkpointing have the lowest total energy consumption as they select mobile devices with low energy consumption rates and do not perform replication. Compared with the proposed algorithm, dynamic grouping uses less energy (about 32%) since it takes advantage of checkpointing for reliable groups.
The percentages of task failures in first sample are illustrated in Fig. (3.d), where the percentage of failure task in this proposed algorithm is lower than others. on average, percentage of failure task in this proposed algorithm is 11% lower than reliable offloading and 6% lower dynamic grouping, because tasks are replicated in several mobile devices.
Second Sample: Here, the aforementioned algorithms are reviewed and compared by taking fail-fast in case2. In this situation, the proposed algorithm applies checkpointing for fault tolerance because subtasks have high computational loads.
The success rate in the second sample is shown in Fig. (4.a), where replication achieves first ranks (12% higher

than the proposed method) given that it replicates tasks in several mobile devices. However, it is not efficient in this sample because subtasks have high computational loads.

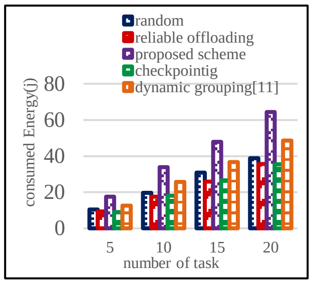
The completion time in second sample is expressed in Fig. (4.b), where the proposed algorithm and has the lowest completion time because failed subtasks resume execution on new service provider devices from the latest recorded checkpoint. As it is illustrated, on average, the completion time in the proposed algorithm by 11%, 16%, 13%, and 27% is lower than reliable offloading, replication, and dynamic grouping, respectively. Thus, proposed algorithm saves completion time. This improvement is not very impressive, however, because of fail-fast.



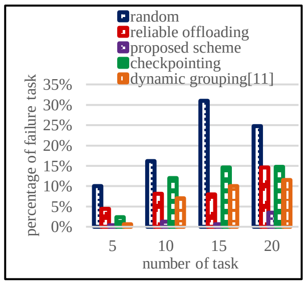
a. Success rate



b. Completion time



c. Consumed Energy



d. Percentage of task failure

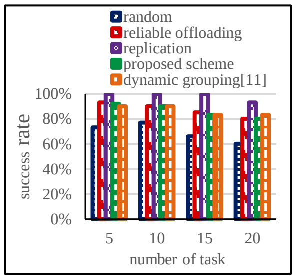
Fig 3. Impact of different numbers of subtasks on the proposed algorithm and previous methods according to fail-fast and subtasks of case1

The success rate in the second sample is shown in Fig. (4.a), where replication achieves first ranks (12% higher than the proposed method) given that it replicates tasks in several mobile devices. However, it is not efficient in this sample because subtasks have high computational loads.

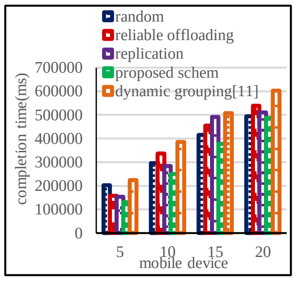
The completion time in second sample is expressed in Fig. (4.b), where the proposed algorithm and has the lowest completion time because failed subtasks resume execution on new service provider devices from the latest recorded checkpoint. As it is illustrated, on average, the completion time in the proposed algorithm by 11%, 16%, 13%, and 27% is lower than reliable offloading, replication, and dynamic grouping, respectively. Thus, proposed algorithm saves completion time. This improvement is not very impressive, however, because of fail-fast.

Fig. (4.c) displays the overall energy consumption of all devices for each algorithm with the second sample. Replication has the highest consumption of energy since tasks are replicated on several mobile devices. This increased energy consumption is impressive because subtasks are large. The proposed algorithm, on the other hand, exhibits the lowest energy consumption, which is 25%, 4.5%, 50%, 37% greater than random, reliable offloading, replication, and dynamic grouping, respectively. The reason is that the new service provider does not need to execute subtask from the beginning, which leads to a reduction of energy requirements.

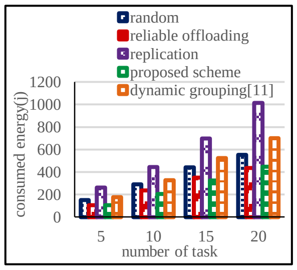
Finally, Fig. (4.d) shows task failure percentages with the second sample. Failure percentage of the proposed algorithm is lower compared to dynamic grouping and random allocation. This can be attributed to the selection of reliable mobile devices.



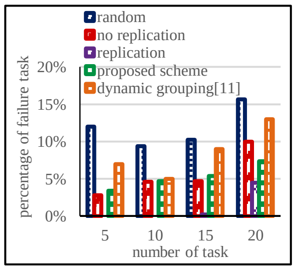
a. Success rate



b. Completion time



c. Consumed Energ



d. Percentage of task failure

Fig 4. Impact of different numbers of subtasks on the proposed algorithm and previous methods according to fail-fast and subtasks of case 2

6.2- Second Scenario

In this scenario, the proposed algorithm is evaluated and compared by considering fail-slow in both cases.

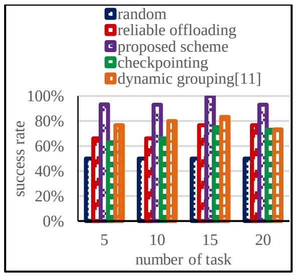
First Sample: The mentioned algorithms are reviewed and compared by considering fail-slow in case1. In this situation, after the proposed algorithm selects reliable mobile devices it applies replication for fault tolerance because subtasks have low computational load.

The success rate with the first sample is shown in Fig. (5.a). The success rate of the proposed algorithm is higher than that of the other algorithms because reliable mobile devices with the highest rank are selected as a resources and subtasks are replicated to several devices. Hence, as it is illustrated, the proposed algorithm with average of 89%, 32%, 36%, and 21% surpasses random, reliable offloading, checkpointing, and dynamic grouping, respectively.

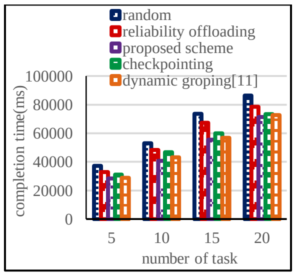
Completion times with the first sample are shown in Fig. (5.b). As evident, the proposed algorithm has the lowest completion. It is expected that checkpointing have the lowest completion time because new service providers do not need to execute failed subtask from the beginning. However, this is not true in this case as, for small tasks, the overhead of getting checkpoints is larger than that of restarting.

The total energy consumption of all devices for each algorithm with the first sample is shown in Fig. (5.c). Once again, the proposed algorithm has the highest consumption since tasks are replicated to several devices, while dynamic grouping consumes lower energy, which exceeds the proposed method by 25%.

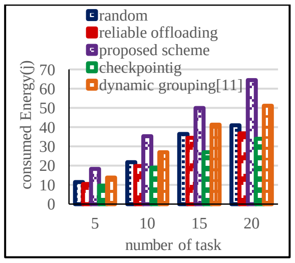
By comparing the algorithms using the first sample of the two scenarios, it can be deduced that success rate of the proposed algorithm is higher than other algorithms. Moreover, the proposed algorithm has the highest energy consumption. This increasing energy consumption is not significant because subtasks are small.



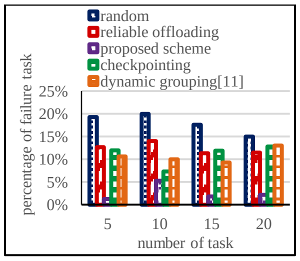
a. Success rate



b. Completion time



c. Consumed Energy



d. Percentage of task failure

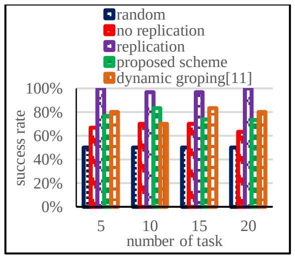
Fig 5. Impact of different numbers of subtasks on the proposed algorithm and previous methods according to fail-slow and subtasks of case1

Second Sample: Here, the mentioned algorithms are reviewed and compared by considering fail-slow in case2. Under these circumstances, after the proposed algorithm selects reliable mobile devices as resources to assign tasks, it applies checkpointing for fault tolerance because subtasks have high computational loads.

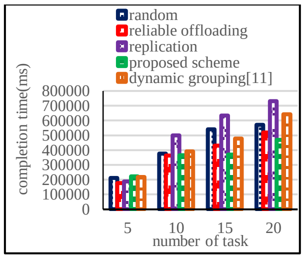
Success rates with the second sample are shown in Fig. (6.a). The success rate of replication is higher than that of the other algorithms (28% higher than the proposed method).

The completion time in the second sample is shown in Fig. (6.b). The completion time of the proposed algorithm is lower than that of the other algorithms (16%, 4%, 30%, and 17% lower than random, reliable offloading, replication, and dynamic grouping, respectively). The improvement in completion time is significant because of fail-slow and tasks with high computational loads. However, when the number of subtasks decreases, checkpointing overhead increases in relation to task size. Therefore, completion time increases**.**

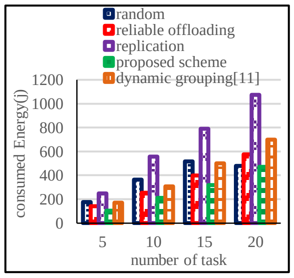
The total energy consumption of the algorithms can be seen in Fig. (6.c). Replication uses the largest amount of energy (59% more than the proposed algorithm). The increase in energy consumption is significant because subtasks are large. The total energy consumption in the proposed algorithm is lower than that of the other algorithms (28%, 19%, and 34% lower than random, reliable offloading, and dynamic grouping, respectively).



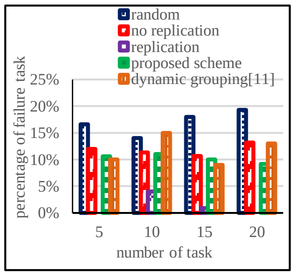
a. Success rate



b. Completion time



c. Consumed Energy



d. Percentage of failure task

Fig 6. Impact of different numbers of subtasks on the proposed algorithm and previous methods according to fail-slow and subtasks of case2

After comparing the algorithms in the second sample of the first and second scenarios, it can be deduced that in the second sample, the total energy consumption and completion time of the proposed algorithm are lower than other algorithms because, in the proposed algorithm, new service providers do not need to execute failed subtask from the beginning. This improvement in second scenario is better than the first because of fail-slow.

These results indicate that replication is very costly for tasks with high computational load because, in replication, energy consumption is high. Although, success rate in checkpointing is lower than that of replication algorithm, checkpointing is more appropriate for tasks with high computational loads. Consequently, in this paper we tried to make a trade of between energy and success rate. As it is illustrated, our algorithm saves impressive amount of energy in tasks with high computational load, while success rate of proposed algorithm is lower than Replication. In contrast, for tasks with low computational load, the success rate of the proposed algorithm is higher than other algorithms, while it has the highest energy consumption. However, this increasing energy consumption is not significant because subtasks are small.

7- Conclusion

In this paper, a new approach was proposed for fault tolerance where a fully distributed resource allocation algorithm was applied without using any central component with the objective to improve the reliability of mobile resources. Mobile devices in this algorithm are adopted as resources to which tasks are assigned. In the context of mobile devices, energy constraints, mobility, and availability are considered as fault factors used to predict device states and prevent faults caused by volatility of mobile devices. The algorithm applied replication or checkpointing for fault tolerance according to the task size. Here, a context-aware reliable offloading middleware was developed to collect contextual information, manage reliable offloading processes, and fault tolerance. To evaluate the proposed method, several experiments were run in a real environment. The results showed higher success rate as well as significant improvements in completion time and energy consumption for tasks with high computational loads.
In future studies, secure offloading by assigning tasks to trusty devices would be of concern to overcome malicious users. Moreover, extending the proposed method for scenarios in which multiple offloading requests are submitted simultaneously is regarded as another future work.

References

[1] N. Fernando, S. W. Loke, and W. Rahayu, “Mobile cloud computing: A survey,” *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84-106, 2013.

[2] M.Othman, S. A. Madani, and S. U. Khan, “A survey of mobile cloud computing application models,” IEEE Communications Surveys & Tutorials, vol. 16, no. 11, pp. 393-413, 2014.

[3] G. F. Huerta Cánepa, “A context-aware application offloading scheme for a mobile peer to peer environment,” Ph.D. dissertation, Department of Information and Communication Engineering, KAIST, South Korea, 2012.

[4] M. Conti, et al. “Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence,” Pervasive and Mobile Computing, vol. 8, no. 1, pp. 2-21, 2012.

[5] V. Cardellini, V. De NitoPersoné, V. Di Valerio, F. Facchinei, V. Grassi, F. Lo Presti and V. Piccialli, “A game-theoretic approach to computation offloading in mobile cloud computing,” Technical Report, 2013..

[6] S. G. Falavarjani, M. Nematbakhsh, and B. S. Ghahfarokhi, "Context-aware multi-objective resource allocation in mobile cloud," Computers & Electrical Engineering,vol. 44, pp. 218-240, 2015.

[7] C.Shi, et al. “Serendipity: enabling remote computing among intermittently connected mobile devices,” In Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing. ACM, 2012, pp. 145-154.

[8] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, “A context sensitive offloading scheme for mobile cloud computing service,” In Proceedings of IEEE of 8th International Conference on In Cloud Computing (CLOUD), 2015, pp.869-876.

[9] D. N. Raju, and V. Saritha. "Architecture for fault tolerance in mobile cloud computing using disease resistance approach." International Journal of Communication Networks and Information Security, vol. 8, no. 2, 2016.

[10] B. Zhou and R. Buyya, "A Group based Fault Tolerant Mechanism for Heterogeneous Mobile Clouds," In Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Nov 2017.

[11] J. Park, H. Yu, H. Kim, and E. Lee, “Dynamic group- based fault tolerance technique for reliable resource management in mobile cloud computing,” Concurrency and Computation: Practice and Experience, John Wiley & Sons, vol. 26, no. 17, Jan. 2014.

[12] J. Park, H. Yu, E. Lee, “Resource allocation techniques based on availability and movement reliability for mobile cloud computing,” Distributed Computing and Internet Technology, Springer Berlin Heidelberg, 2012,pp. 263–264.

[13] J. S. Park and E. Y. Lee, “Entropy-based grouping techniques for resource management in mobile cloud computing,” Ubiquitous Information Technologies and Applications, Springer Netherlands, 2013, pp. 773-780.

[14] P. Stahl, et al, "Dynamic Fault-Tolerance and Mobility Provisioning for Services on Mobile Cloud Platforms." In Proceedings of the 2017 5th International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), IEEE, 2017, pp. 131-138.

[15] S. Choi, K. Chung, and H. Yu, “Fault tolerance and QoS scheduling using CAN in mobile social cloud computing”, Cluster Computing, vol.17, no.3, pp. 911-926, 2014.

[16] E. E. Marinelli, Hyrax: cloud computing on mobile devices using MapReduce. No. CMU-CS-09-164. Carnegie-mellon univ Pittsburgh PA school of computer science, 2009.

[17] C-A. Chen, et al., “Energy-efficient fault-tolerant data storage and processing in mobile cloud,” IEEE Transactions on cloud computing, vol. 3, no. 1, pp. 28-41, 2015.

[18] J. Park, H. Yu, K. Chung, and E. Lee, “Markov Chain based Monitoring Service for Fault Tolerance in Mobile Cloud Computing,” In Proceedings of IEEE Workshops of International Conference on Advanced Information Networking and Applications, 2011, pp.520-525.

[19] P. Patel and V. Prakash, “FTAB: Fault Tolerance Approach by Using HMM with BAUM-WELCH Algorithm in MCC,” In Proceedings of Tenth international conference on Wireless and Optical Communication Network (WOCN), 2013, pp.1-4.

[20] L. Ling, W. Zhulin and Y. Xiuhua, “Mobile Resource Reliability-Based Task Allocation for Mobile Cloud”, In Proceedings of the 2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC), IEEE, 2015, pp.1746-1750.

[21] J.Park, et al, “Two- phase grouping- based resource management for big data processing in mobile cloud computing,” International Journal of Communication Systems, vol. 27, no. 6, pp. 839-851, 2014.

[22] L. McNamara, C. Mascolo, L. Capra, “Media sharing based on colocation prediction in urban transport,” In Proceedings of the 14th ACM international conference on mobile computing and networking, ACM, 2008. pp. 58–69.

[23] P. A, Lee, and T. Anderson, “Dependable computing and fault tolerant systems,” Fault Tolerance: Principles and Practice, Springer Verlag, NewYork, vol. 3, 1990.

[24] Ch. Song, et al. “Limits of predictability in human mobility,” Science 327.5968 , 2010, pp.1018-1021.

[25] J. Nicholson and B. D. Noble. “Breadcrumbs: forecasting mobile connectivity,” In Proceedings of the 14th ACM international

conference on Mobile computing and networking. ACM, 2008, pp. 46-57.
[26] Huang, Wei, et al. “Predicting human mobility with activity changes,” International Journal of Geographical Information Science, vol. 29, no. 9, pp. 1569-1587, 2015.
[27] S. Ch. Shah, “Energy efficient and robust allocation of interdependent tasks on mobile ad hoc computational grid,” Concurrency and Computation: Practice and Experience, vol. 27, no. 5, pp. 1226-1254, 2015.
[28] M.Sepahkar and M. R. Khayyambashi, “A novel collaborative approach for location prediction in mobile networks,” Wireless Networks, pp. 1-12, DOI 10.1007/s11276-016-1304-1, 2016.
[29] J. A. Gubner, Probability and random processes for electrical and computer engineers, Cambridge University Press, 2006.
[30] M. Wiesmann, et al. “Understanding replication in databases and distributed systems”, In Proceedings of the 20th International Conference on IEEE, Distributed Computing Systems, 2000, pp. 464-474.
[31] R. Tuli and P. Kumar, “Analysis of recent checkpointing techniques for mobile computing systems,” International Journal of Computer Science & Engineering Survey, vol. 2, no. 3, 2011.
[32] E. Cuervo, et al. “MAUI: making smartphones last longer with code offload,” In: Proceedings of the 8th international conference on mobile systems, applications, and services, MobiSys’10, 2010, pp. 49–62.
[33] M.D. Kristensen, “Scavenger: transparent development of efficient cyber foraging applications,” In: Proceedings of the IEEE international conference on pervasive computing and communications (PerCom); 2010. p. 217–26.
[34] C. L. Hwang, and K. Yoon. Multiple attribute decision making: methods and applications a state-of-the-art survey, Springer Science & Business Media, Vol. 186, 2012.
[35] C. E. Shannon, “A mathematical theory of communication,” ACM SIGMOBILE Mobile Computing and Communications Review, vol. 5, no. 1, pp. 3-55, 2001.
[36] FaceDetector, http://developer.android.com/reference/android/media/FaceDetector.html Avalaible Online @ September 2016.
[37] L. Zhang, et al. “Accurate online power estimation and automatic battery behavior based power model generation for smartphones,” In Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. ACM, 2010, pp. 105-

Zahra Najafabadi Samani received her B.Sc. degree in Computer Hardware Engineering from Azad University, Najafabad Branch, Iran in 2010, and M.Sc. degree in Computer Architecture from University of Isfahan, Iran, in 2016. Her research interests include Distributed Systems and High Performance Computing, Cloud and Mobile Cloud Computing, Optimization and multi-criteria decision analysis method, Future Internet Network Architectures.

Mohammad Reza Khayyambashi received his B.Sc. degree in Computer Hardware Engineering from Tehran University, Tehran, Iran in 1987. He received his M.Sc. in Computer Architecture from Sharif University of Technology (SUT), Tehran, Iran in 1990. He got his Ph.D. in Computer Engineering, Distributed Systems from University of Newcastle upon Tyne, Newcastle upon Tyne, England in 2006, he was research assistant during his Ph.D. course at University of Newcastle upon Tyne, Newcastle upon Tyne, England. He is now working as an Associate Professor at the Department of Computer Architecture, Faculty of Computing Engineering, University of Isfahan, Isfahan, Iran. His research interests include Distributed Systems, Computer Networking, Mobile Cloud, Cloud Computing, Software Define Network (SDN), Mobile and Social Networks.