A Group-based Fault Tolerant Mechanism for Heterogeneous Mobile Clouds

Bowen Zhou
The Cloud Computing and Distributed Systems Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia
bowenz@student.unimelb.edu.au

Rajkumar Buyya
The Cloud Computing and Distributed Systems Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia
rbuyya@unimelb.edu.au

ABSTRACT

Recent developments of the enhanced hardware on mobile devices such as quad-core CPUs and various sensors have made it possible to build a powerful heterogeneous mobile cloud offloading service that consists of a mobile ad-hoc cloud, nearby servers and public cloud services. However, the availability and mobility management of mobile devices in the network can significantly hinder the performance of mobile cloud systems due to the frequent system faults caused by dynamic changes, and prevent applications from offloading to mobile ad-hoc networks. In order to improve the mobile cloud service reliability, we propose a group based fault tolerant mechanism GFT-mCloud that classifies mobile devices into groups based on its processing capacity, mobility, and reliability. Different fault tolerance techniques are then devised adaptively based on the task offloading schedules and the specific group of machines it's offloaded. GFT-mCloud is designed as a standalone module that can work with existing mobile cloud code offloading systems. Extensive experiments have been conducted to evaluate the proposed mechanism. The results show that our fault tolerant mechanism is able to outperform conventional fault tolerant algorithms in the mobile cloud offloading environment.

CCS CONCEPTS

•Computer systems organization →Cloud computing; •Human-centered computing →Mobile computing; •Networks →*Network reliability;*

KEYWORDS

Mobile cloud computing, fault tolerance, code offloading

ACM Reference format:
Bowen Zhou and Rajkumar Buyya. 2017. 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, Melbourne, VIC, Australia, November 7–10, 2017 (MobiQuitous 2017),* 10 pages.
DOI: 10.1145/3144457.3144473

MobiQuitous 2017, Melbourne, VIC, Australia

DOI: 10.1145/3144457.3144473

1 INTRODUCTION

Recent years have seen the exponential development of smart mobile devices. The powerful processors, higher capacity memory, and various sensors introduce opportunities for new types of cognitive mobile applications such as personal health monitoring, panorama videos, and augmented reality on mobile devices. These emerging mobile applications rely on intensive sensor data processing, which is a non-trivial task on a single mobile device due to its constrained resources.

Mobile cloud computing (MCC) comes into sight as a new computing paradigm that leverages cloud resources to enhance the performance of mobile devices. The term "cloud" here refers to a shared resource pool consisting of both mobile devices cloud (i.e. mobile ad-hoc networks) [8], nearby cloudlets [20], and private/public cloud services [27]. The shared resources collaborate with each other to process the data obtained from sensors on each mobile device. The paradigm enables mobile devices to utilize computation offloading as well as crowd sourcing among the shared computing resources to enhance the performance of cognitive mobile applications. However, as a resource sharing environment of heterogeneous devices connected via wireless communication mediums, the reliability of the machines and the computing environment have significant effect on the system performance.

Since mobile cloud systems consist of mobile devices and servers connected via wireless networks, the mobility of the devices and intermittent network connections have become the major issues that induce system failures and further hinder the system reliability. Many fault tolerance solutions have been proposed to tackle the reliability issue in distributed computing systems, e.g., computer grids, mobile grids, and cloud computing systems. These approaches mostly consider a homogeneous computing environment composed of identical machines and wired network connections. As a result, the fault tolerant policies proposed are often onefold solutions that lack adaptivity to the heterogeneous computing environment. On the contrary, mobile clouds are often composed of devices and wireless networks that have different specifications such as CPU speed, data throughput and signal strength. Therefore, existing fault tolerance approaches may not be able to maintain a similar system reliability level on mobile cloud systems as they are on grid computing systems.

In mobile cloud computing, code offloading and task delegation are two main techniques for resource augmentation. Many algorithms and frameworks have been proposed to make task offloading decisions for a variety of mobile cloud computing environments [1, 4, 9, 18]. However, these approaches mainly focus on the performance gain of task offloading without much development in

improving the reliability of task offloading execution in mobile clouds. In this paper, we aim to propose a standalone fault tolerant mechanism that can work with existing frameworks and utilizes task offloading schedules devised from existing mobile code offloading algorithms to maintain the reliability of mobile cloud systems.

In order to make the FT mechanism adaptive to the heterogeneous mobile cloud, we propose a group-based fault tolerant mechanism GFT-mCloud for mobile cloud systems using reactive fault tolerant techniques. The machines such as mobile devices, cloudlets and public cloud servers are classified into different groups based on their hardware properties such as processing speed, reliability, and battery lifetime. The proposed FT mechanism can dynamically adjust different fault tolerant policies for different resource groups as well as different critical level of mobile tasks in a mobile application to improve the system reliability with lower processing overhead from the redundancy incurred.

The key contributions of this paper are as follows.

- We present a machine grouping algorithm for dynamically classifying machines in mobile cloud systems into multiple groups that have machines with similar capabilities. The groups can then be utilized by the proposed FT mechanism to obtain customized fault tolerant policies. The machine grouping algorithm is designed to be expandable for more grouping criteria.
- We propose a standalone fault tolerant mechanism GFT-mCloud for mobile cloud systems that can work with mobile code offloading algorithms. It is able to devise different fault tolerant policies adaptively for different machine groups with the consideration of minimizing the overhead generated by the redundancy.
- The proposed fault tolerant mechanism is implemented as a library that can be added into the existing Android mobile cloud offloading frameworks as well as simulation environments.
- We implement a simulation environment of heterogeneous mobile clouds based on BRITE [14] and Weka, and test the proposed algorithm with various experiment scenarios as well as the efficiency of working with existing code offloadng algorithms in mobile cloud computing.

The rest of this paper is organized as follows. Section 2 discusses the related works on the fault tolerant approaches in grid computing, mobile ad-hoc networks, and mobile clouds. Then we describe the heterogeneous mobile cloud environment (HMC) and present the preliminaries and models in Section 3. With the help of the models, we propose the group-based fault tolerant scheduling algorithm for HMC in Section 4. Section 5 presents the evaluation and discussions on the experimental results. Finally, conclusions and future work are discussed in Seciton 6.

2 RELATED WORK

2.1 Task offloading in mobile cloud computing

Task offloading algorithms in mobile clouds aim to make decisions on whether, where and how to offload mobile tasks to other computing resources. The computing resources can be public cloud, nearby cloudlets, and mobile devices. Task offloading algorithms take into consideration task requirements such as computation and deadline, and machine capabilities to decide the task execution location (local or offloading) to obtain system gains (shorter execution time or less energy consumption). Cuervo et al. [4] proposed a code offloading framework for .NET applications. Linear programming optimization framework is applied to decide which module of an application to offload. ThinkAir [9] is an Android platform for code offloading that aims to lower the execution time and energy consumption at the same time. Rahimi et al. [18] proposed a multi-tier offloading framework that uses a greedy heuristic to make the offloading decisions considering device mobility. Only few works have applied group-based task offloading techniques, and focus on the group of tasks rather than devices. Xiang et al. [25] proposed a coalesced offloading method to combine tasks from multiple applications and offload together to minimize the energy consumption.

2.2 Fault tolerance techniques

Checkpointing and replication are two commonly used reactive fault tolerance techniques [22]. Checkpointing periodically takes snapshots of applications that contain running states and saves snapshots on reliable storage. The time between checkpoints depends on the system reliability. On the contrary, replication does not require state saving. Instead, it runs the replication of an application simultaneously on multiple computing resources to ensure task complete execution percentage meets certain level.

2.2.1 Fault tolerance in distributed computing. Distributed computing paradigms include cluster computing, grid computing, mobile grids, etc. The loosely coupled computing resources connected via networks require fault tolerance management to maintain the system reliability. Dobber et al. [5] compared the performance of job replication (JB) and dynamic load balancing (DLB) on distributed system robust level. They provided a threshold value Y of Y^* based on job execution time that JB outperformed DLB when $Y < Y^*$. Naksinehaboon et al. [15] proposed an incremental checkpoint and restart model for high performance computing (HPC). To reduce the overhead of checkpointing, the model aims to perform a set of incremental checkpoints between two full checkpoints by only saving address space that has changed since the last checkpoint. Noticeably, the fault tolerance in distributed computing systems only considers machine crash failures since the wired networks are stable. However, for a wireless computing system such as mobile cloud, device mobility and network link stability are of concern. Litke et al. [12] presented a replication-based algorithm, which utilizes the Weibull distribution for mobility analysis to estimate the number of replicas in order to maintain a certain level of fault tolerance for mobile grids. Most of the existing algorithms only apply the onefold fault tolerance policy, which is impractical for resource-scarce mobile devices involved computing systems as there is no guarantee for available computing nodes.

2.2.2 Fault tolerance in mobile cloud computing. A few fault tolerance algorithms were proposed for mobile cloud computing based on grouping. Chen et al. [2] proposed the k-out-of-n reliability control method to achieve energy efficiency and maintain system reliability level for data storage and processing in mobile cloud. The failure probability of a node is estimated by three factors: remaining battery, node mobility, and application factor. Park

et al. [16] presented a fault tolerant algorithm for mobile cloud resource management. Different fault tolerance techniques such as checkpoint and replication are applied to different groups based on its availability and mobility. However, the grouping algorithm only considers device mobility and utilization rate. It is inextensible for more criteria. Choi et al. [3] classified devices in mobile grids into groups based on similar hardware properties. For tasks dispatched to low reliability groups, task replication is applied for fault tolerance, whereas the tasks dispatched to high reliability groups are migrated to another device upon failures.

3 PRELIMINARY MODELS

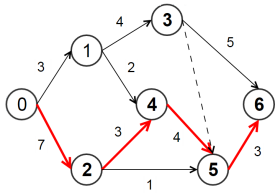
In this section, we formally present the models of the mobile cloud system, including application models, machine models, and mobility model. We consider the mobile cloud computing system as a network of heterogeneous machines consisting of mobile devices, nearby cloudlets, and public cloud VMs. The machines are loosely connected via various types of wireless networks such as Bluetooth, mobile cellular and WiFi. The mobile devices are free to join and leave the network at any time while the cloudlets and public cloud VMs are considered stable. Therefore, the failures considered by this work are: 1) Machine crash failures, where machines either halt or do not response; 2) Loss of connection failures, where either wireless connections are disrupted or machines move out of the range of the mobile cloud network. The Byzantine failures [11] are not the focus of this work.

3.1 Application model

Each mobile device in the network runs a mobile application where its tasks are evaluated to either offload to another machine or not. An application is modelled as a directed acyclic graph (DAG) $A = \langle S, E \rangle$ (example shown in Figure 1), where S represents the set of tasks s_i within the application, and E represents the dependencies of the tasks. The weight of each edge denotes the execution time of its pointed task. For instance, task s_1 in Figure 1 takes 3 time units to complete after task s_0 completed. It is assumed a task cannot be divided into subtasks and that it needs to be executed as a whole on a single processor of the machine. Each task is associated with a time variable $T_{tf}^{s_i}$ called total float, which is the amount of time a task can be delayed without delaying the completion of the entire application. $T_{tf}^{s_i}$ of task s_i can be obtained by calculating the difference between its latest finish time and earliest finish time. Then the critical path of a DAG can be identified as a path of tasks with 0 total float, which indicates the longest task execution time, i.e., the earliest finishing time of the application [7]. The rest of tasks can be finished as late as its total float and will not affect the completion time of the application. Therefore, tasks on critical paths of an application should be applied with fault tolerant policies to ensure the application to complete on time.

3.2 Machine model

We consider the mobile cloud computing infrastructure M as a network of n heterogeneous machines. $m_i \in M (i = 1, 2, \cdots, n)$ denotes the ith machine. The term "machine" refers to either mobile devices, cloudlets, or cloud instances. We model the hardware specifications of a machine as follows.



Task	Dependency
1	0
2	0
3	1
4	1, 2
5	2, 3, 4
6	3, 5

Figure 1: An example of application model. Critical path is marked in red

$$m_i \triangleq < \mu_i, \theta_i, I_i^{wifi}, I_i^{cell}, I_i^{bt}, B_i^{wifi}, B_i^{cell}, B_i^{bt}, T_i^{avail}, T_i^r > \quad (1)$$

μ_i represents the processing speed of the machine. θ_i represents the utilization rate of the processor. $I_i^{wifi}, I_i^{cell}, I_i^{bt}$ are binary indicators denoting if the machine is equipped with that type wireless interface. For instance, if $I_i^{wifi} = 1$, machine m_i has access to WiFi, otherwise if $I_i^{wifi} = 0$. Variables $B_i^{wifi}, B_i^{cell}, B_i^{bt}$ represents the bandwidth of the wireless medium on machine m_i. The network speed of each wireless medium is the product of bandwidth and binary indicators I_i, plus current network latency. T_i^{avail} is the earliest available time of machine m_i, which represents the current workload of the machine. T_i^r denotes the time between failures of machine m_i. It is obtained by modelling the mobility of the machine.

3.3 Mobility model

In order to take into account the machine reliability, the available time of a machine is considered. Note that the study of mobility model and trajectory is not the focus of this paper.

To obtain the available time of a machine, the mean time between failures (MTBF) is adopted as T_i^r. MTBF describes the expected time of a machine between failures. In this case, it represents the expected operating time of a machine before it is disconnected from the network. The Weibull distribution is often applied to effectively represent the machine availability in distributed computing environments [19]. The MTBF of a machine can be calculated from the Weibull distribution that abstracts the machine's behaviour. The probability density function of a 2-parameter Weibull distribution is given by:

$$f(t) = \frac{\beta}{\eta}(\frac{t}{\eta})^{\beta-1} e^{-(\frac{t}{\eta})^{\beta}}. \quad (2)$$

The history of the machine connection time to the mobile cloud network is used to estimate the shape parameter β and scale parameter η. It consists of records of the time of that machine established a wireless connection as well as the duration of that connection.

The estimation of Weibull distribution parameters are obtained by applying the linear regression method to the cumulative distribution function (CDF) of Weibull distribution:

$$F(x) = 1 - e^{-(\frac{x}{\eta})^{\beta}}. \quad (3)$$

Then Equation 3 is converted into a linear equation form as

$$ln(ln(\frac{1}{1 - F(t)})) = \beta ln(t) - \beta ln(\eta). \quad (4)$$

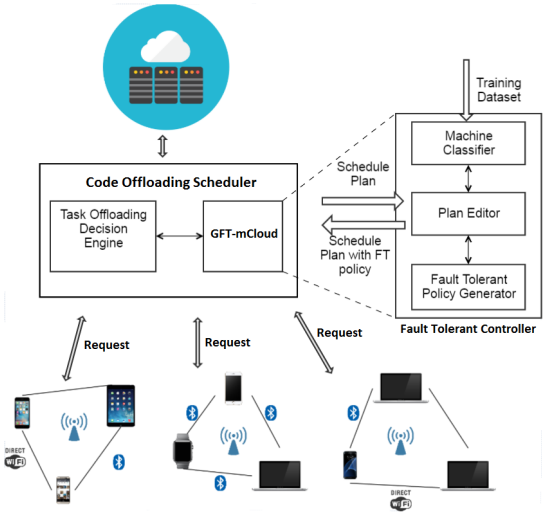


Figure 2: The proposed fault tolerant mechanism as a standalone module

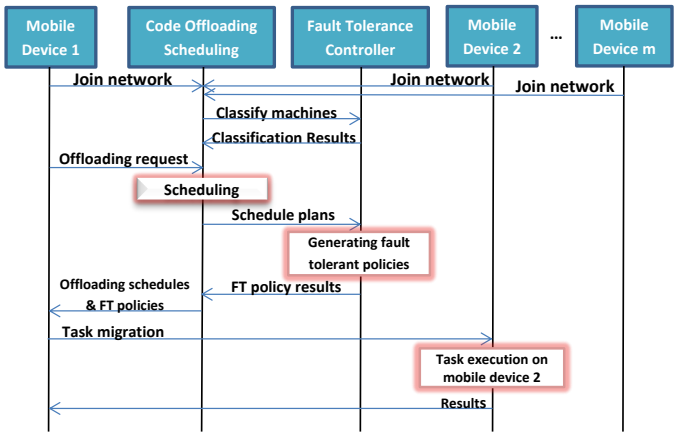


Figure 3: Interaction procedures of proposed FT mechanism

Then the parameters can be calculated using median ranks and least-squares fit on the data points generated from the history of the machine connection time in the network.

4 GROUP-BASED MOBILE CLOUD FAULT TOLERANT MECHANISM GFT-MCLOUD

The mobile cloud computing environment usually comprises heterogeneous mobile devices, cloud VMs, and intermittent wireless network connections. To cope with the dynamics and heterogeneity, in this paper, we propose a group-based fault tolerant mechanism GFT-mCloud that classifies machines into different groups based on its capability and adjusts different fault tolerance policies for tasks offloaded to different groups of machines.

The GFT-mCloud is designed to operate on the offloading schedules of existing task offloading frameworks in mobile cloud computing. The position of the proposed standalone fault tolerant mechanism with the existing mobile cloud offloading frameworks is shown in Figure 2 and the proposed fault tolerant policies generating procedures in the view of one of the mobile devices are depicted in Figure 3. All the mobile devices follow the same procedures. The Code Offloading Scheduler is provided by the existing mobile cloud frameworks that have a decision engine to schedule the offloading tasks. We name the outputs of the decision engine as schedule plans. GFT-mCloud works as an add-on module to the framework. It includes a *Machine Classifier* for grouping machines, a *Policy Generator* for devising fault tolerant policy for each task, and a *Plan Editor* to decode the offloading schedule plan and encode the fault tolerant policy generated.

4.1 Machine grouping algorithm

The grouping algorithm considers three criteria: machine processing capability, machine availability, and communication condition of the machine. In order to provide an automatic classification approach that can adopt an arbitrary number of criteria, clustering and decision tree learning are adopted in the machine grouping algorithm. Decision tree [6] is a tree-like graph that is comprised of interior nodes each representing a property variable, and leaf nodes each representing a class label. The path from the root node to each leaf node represents the values of the properties that classify the machines.

For each interior node, there is a set of split values that divide the set of objects into different groups. Moreover, the property variable on each level of the decision tree needs to be selected in order to provide an accurate classification. The decision tree structure can be trained from a set of supervised object data with the property values and its class label. The machine grouping algorithm includes two phases: data pre-processing phase and decision tree generating phase.

4.1.1 Data pre-processing. First, the data set of machine instances with property values needs to be labelled so that it can be used to train the decision tree. The k-means clustering method [13] is used for labelling the data set with sub-classes based on each of the three criteria. The three criteria are further set with sub-classes based on the characteristics of the mobile cloud system. The machine processing capability criteria (**P**) is divided into slow, medium and high processing speed. The machine availability criteria (**A**) is divided into low and high since the machines in mobile cloud systems are mostly mobile devices and stationary machines. The communication condition criteria (**C**) is divided into poor and good group. The types of criteria and number of sub-classes of criteria can be set by users in order to cooperate with the requirements of different executing conditions. In our case, a total of 12 labels are considered.

Second, each criteria needs to be quantified based on the property values from the machine data set. The processed data set is named as *learning set*. The processing capability is quantified by calculating the CPU speed of the machine and its utilization as follows.

$$P_i = \mu_i * \theta_i, \tag{5}$$

where P_i is the processing capability of machine m_i, μ_i is the CPU speed, and θ_i is the utilization of the CPU.

For machine availability, the algorithm considers the device mobility and the remaining battery lifetime of the device. Based on the mobility model described in Section 3, the device mobility is represented by its available time in the mobile cloud network. Since the longer the available time of device there is and the more battery there remains, the higher its availability is, the availability is defined as follows.

$$A_i = \alpha_{avail} T_i^r + \beta_{avail} \rho_{battery}, \tag{6}$$

where T_i^r is the device available time obtained from the mobility model, and $\rho_{battery}$ and the remaining battery percentage. $\alpha_{avail}, \beta_{avail}$ are weight factors that can be adjusted based on user preferences, and $\alpha_{avail} + \beta_{avail} = 1$. The two components of the equation are normalized to a range of 0 to 1.

The communication condition reflects on the data throughput of the wireless networks in mobile cloud systems. For both checkpointing and replication policy GFT-mCloud considers, it requires fast communication and low energy consumption to constrain the incurred overhead. Therefore, the direct connections to other machines as well as the speed of the wireless connection are both considered. The communication condition is quantified as follows.

$$C_i = \sum_w (\alpha_{conn} B_i^w + \beta_{conn} * \frac{k^w}{n}) * I_i^w, \tag{7}$$

where B_i^w is bandwidth of wireless medium w, k^w is number of machines that machine m_i directly connected to via wireless medium w, and n is the total number of machines in its current mobile cloud network. The bandwidth of the wireless link B_i^w explicitly represents the data transmission speed. k^w reflects the level of redundancy machine m_i can achieve in the network as the more machines it has direct connections to, the more machines can be chosen for its fault tolerance redundancy. $\alpha_{conn}, \beta_{conn}$ are the weight factors that can be adjusted by the system, and $\alpha_{conn} + \beta_{conn} = 1$. The two components of the equation are normalized to a range of 0 to 1.

4.1.2 Machine grouping algorithm. The machine grouping algorithm operates on the *learning set* with ID3 (Iterative Dichotomiser 3) algorithm [17] to generate the decision tree for further machine grouping. The pesudocode is shown in Algorithm 1.

S represents the *learning set*, C is the set of criteria (in this case, three criteria available), V denotes the set of split values for each criteria, and L is the set of class labels. Algorithm1 returns the decision tree T for machine grouping. The algorithm first creates a root node for the tree. Then the criterion with the maximum entropy value is selected as the criteria for the root node (step 1-7). The entropy is used to evaluate differences between groups and is calculate as $entropy(S, a_i) = \sum_{v \in V_{a_i}} -v log_2(v)$. The items in S are separated into sub-groups V_j based on split values V_{a^*} of criteria a^* (step 8). For sub-groups V_j (step 9-18), if it is empty then marked as an end node; If it only contains items of label l_n, then make a node of this branch as label l_n; Otherwise, remove criteria a^* from A and run $ID3$ with the sub-group S_{v_j} iteratively until the items in S are classified. The decision tree generated by Algorithm 1 can be updated with more machine data obtained or new criteria need to be added.

Algorithm 1 Generate Decision Tree

1: **procedure** GetDecisionTree(S, C, V, L)
2: $n \leftarrow$ *create a root node for the tree*
3: **for all** $a_i \in A$ **do**
4: $en_i = entropy(S, a_i)$
5: $Gain(a_i) = entropy(S) - \sum_{v \in V} \frac{|S_v|}{|S|} en_i$
6: $a^* \leftarrow MAX(Gain(a_i))$
7: *assign attribute* a^* *to node* n
8: $S_{v_j} \leftarrow$ *seperate* S *based on values in* V_{a^*}, $v_j \in V_{a^*}$
9: **for all** S_{v_j} **do**
10: **if** S_{v_j} *is empty* **then**
11: *mark* S_{v_j} *as a leaf node and end the branch*
12: **else if** S_{v_j} *only contains items of label* $l_n \in L$ **then**
13: *put* l_n *as the leaf node and end the branch*
14: **else**
15: $A = A - a^*$
16: $ID3(S_{v_j}, A, V)$
17: *return* $T \leftarrow ID3(S, A, V)$

4.2 Group-based fault tolerant algorithm

Due to the heterogeneity of mobile cloud system, a onefold fault tolerant policy approach may not adapt different execution conditions. For instance, applying replication method to the group with high reliability machines incur more operating overhead than checkpointing. On the contrary, applying checkpointing to low reliability machines can introduce unnecessary storing snapshots of checkpoints. Moreover, as the task offloading decision modules in mobile cloud systems have already devised an offloading schedule for mobile tasks, the fault tolerance algorithm needs to consider the execution location of the task being offloaded to maintain the offloading benefits as well as load balance.

In order to tackle this issue, we propose a group-based fault tolerance algorithm that takes into consideration the properties of different machine groups and adaptively select either checkpointing or replication as fault tolerant policy for the task based on the group of the machine the task is offloaded onto. The proposed fault tolerant mechanism operates on the task offloading schedules made by the mobile cloud frameworks in order to make the mechanism adaptive to the existing mobile cloud frameworks. Furthermore, the mobile tasks of different mobile applications on the devices are differentiated if they are on the critical path in order to adjust the fault tolerance policies.

Since the proposed fault tolerance mechanism adaptively applies checkpointing and replication methods, the number of replications as well as the frequency of checkpointing need to be determined.

- **Replication**

  The task replication is sent to machines in the same group of the original scheduled machine where the task is offloaded in order to maintain the offloading benefit in terms of execution time and energy consumption. Too many replications can incur large resource redundancy with high energy overhead. To overcome this issue, the machine for replicas is selected by ranking the machines in the group by reliability. The machines are ranked based on the failure rate, the execution time for the replication

Algorithm 2 Group-based fault tolerance algorithm

```
1:  procedure GroupbasedFT(TR, M, J)
2:      if TR needs to update then
3:          TR ← ID3(S, A, V)
4:      for all m ∈ M do
5:          G_TR ← Classify(TR, M)
6:      {C, UC} ← classify the critical path of tasks inJ
7:      while s ∈ J is scheduled by the offloading algorithm do
8:          g ← G_TR(s)
9:          if s ∈ UC then
10:             if T^s_comp < T^s_tf then
11:                 resend task s to originally scheduled machine
12:                 if machine is not available then
13:                     send s to the machine with EFT in g
14:             else
15:                 goto 7
16:         else if s ∈ C then
17:             if g contains L reliabiltiy property then
18:                 calculate ranks of machines in group g
19:                 m* ← lowest ranking machine
20:                 m* ← s_r
21:             else if g contains H reliabiltiy property then
22:                 T_c ← calculate checkpoint frequency
23:                 s ← insert checkpoint with T_c
```

and number of directly connected machines.

$$rank(m) = \alpha T^m_{comp} + \beta p^m_{fail} + \gamma \frac{N_{conn}}{N_{group}},\ \alpha + \beta + \gamma = 1, \quad (8)$$

where α, β, γ are weight factors can be adjusted by users, T^m_{comp} is the task completion time on machine m, N_{conn} denotes the number of machines in the group that machine m has direct connections with, N_{group} is the total number of machines in the current group, and p^m_{fail} is the task execution failure rate calculated as:

$$p^m_{fail} = \frac{N^m_{fail}}{N^m_{total}}, \quad (9)$$

and N^m_{fail}, N^m_{total} are the number of failed tasks and the number of total tasks executed on machine m.

Then the machine with the lowest ranking score is considered the most reliable machine in the group and selected to deploy the replica of the task.

- **Checkpointing** The checkpointing method saves a snapshot of the running process periodically. The frequency of checkpoints is critical since it generates extra network traffic for the wireless communications and the checkpointing also incurs time overhead. Hence, the frequency needs to be determined based on the failure rate and the time taken by checkpointing. The method proposed in Young [26] is adopted to decide the frequency of checkpointing as follows.

$$T_c = \sqrt{2 T_s T_f}, \quad (10)$$

where T_c is the time interval between checkpoints, T_s is the time of checkpointing, and T_f is the time between failures.

The proposed fault tolerant algorithm is listed in Algorithm 2. TR represents the decision tree generated by Algorithm 1, M is the set of machines available in the mobile cloud network, and J is the set of tasks generated from mobile applications running on the mobile devices in M. First, TR is updated if more machine observations are captured in the system (step 2-3). Then, all the machines in M are classified into different groups through decision tree TR (step 4-5). Moreover, the tasks from each application are divided into critical and uncritical tasks using critical path analysis (step 6). After the machines and tasks are classified, the fault tolerance scheduler waits for an offloading decision of a task made by the offloading decision module of the system. Upon receiving the task with the offloading decision, the proper fault tolerance policy is selected (step 7-23). First, the group label g of the machine that task s scheduled is identified (step 8), then the task is identified whether it is a task on the critical path of its application. If it is uncritical and the total float T^s_{tf} is long enough for re-execution (step 10), the task will be resent to the machine scheduled by the offloading decision module upon receiving a failure occurrence. When the machine is not available, the task will be scheduled to the machine within the same group that has the earlier finishing time (EFT) for the task (step 9-15). If the task is on its application's critical path, the fault tolerance policy applied is based on the group of the machine that the task is scheduled. If the group is with **L** reliability property, replication is applied. The replica of tasks is sent to the machine with the highest reliability ranking (step 17-20). If the group is with **H** reliability property, checkpointing is applied. The checkpoint frequency is calculated based on Equation 10 (step 21-23).

The time complexity analysis of GFT-mCloud. Assume there are m machines in the network, and n tasks pass through the fault tolerance algorithm. The machine classification (step 4-5) takes $O(m)$ for grouping. For the fault tolerance policy selection part, the checkpoint frequency calculation (step 21-23) takes $O(1)$ and machine ranking calculation for replication costs $O(m)$. Therefore, the worst case scenario for selecting fault tolerance policy for n tasks costs $O(mn)$. Hence, the overall time complexity of the proposed algorithm is $O(m + mn)$.

5 PERFORMANCE EVALUATION

In order to evaluate the performance of the proposed fault tolerant mechanism, a series of simulations are conducted. The application completion time (i.e. makespan), the mechanism running overhead, and the number of control messages for fault tolerance are the three metrics for evaluation. Results from the proposed algorithm are then compared with other conventional fault tolerance algorithms.

5.1 Experimental Setup

We implement a simulator based on BRITE [14] to simulate the mobile cloud networks. BRITE is able to generate realistic Internet topologies with different bandwidth and delay values for each link of the topologies. We also adopted Weka [24] in the simulation to implement the machine learning methods in the proposed grouping algorithm. Table 1 lists the configuration for the simulation.

Due to the lack of real-world traces that are suitable for the heterogeneous mobile cloud environment described above, the workloads used in the simulation are obtained by profiling an OCR application and a chess game application running on mobile devices in our experimental environment. The workload contains a

Table 1: The configuration for simulations

Workload	**Application:** Optical Character Recognition, Chess game **Number of applications:** 50 **Computation length (No. of instructions):** Uniform(50000, 100000) **Data size (MB):** Uniform(0.5, 10)
Machine	**Processing Speed (MIPS):** Uniform(5000,200000) **Total available time (s):** Uniform(1000,30000) **Failure time point:** Weibull distribution($\alpha = 1.25$, $\beta = 92.74$) **WiFi bandwidth(MBps):** Uniform(1.0,1.2) **Bluetooth bandwidth(MBps):** Uniform(0.2,0.3) **Cellular bandwidth(MBps):** Uniform(0.8,0.9) **Number of machines:** Uniform(20,50)

Table 2: Results of the cross validation

Item	Synthetic Set	CRAWDAD Set 1	CRAWDAD Set 2	CRAWDAD Set 3
Correctly Classified Instances	997	1397	3986	3575
Incorrectly Classified Instances	5	3	14	7
Mean absolute error	0.0008	0.0004	0.0009	0.0003
Relative absolute error	0.6503%	0.2777%	0.6847%	0.2133%
Total number of instances	1002	1400	4000	3582
Accuracy	99.501%	99.786%	99.649%	99.805%

set of applications represented by randomly generated DAGs. Each vertex in the DAG represents a task of the application, and has two values, the amount of computation and data size. The two values are generated from uniform distribution with the range obtained from the application profiling.

The parameters used to characterize the machines in the experimental environment are also listed in Table 1. MIPS (Million Instructions per Second) is adopted in the simulation to represent the processing speed of the machine. It can be profiled by running a multi-thread integer computation looping program with a number of instructions and estimating the elapsed time upon completion. In order to simulate the machine failures, a 2-parameter Weibulll distribution is used for randomly generating the time between failures, and the failure time points are calculated by appending the time generated from the distribution to the machine start time. To capture the mobility patterns of real mobile devices in an area, the CRAWDAD trace sets [10] are used to obtain the shape and slop parameters of the Weibull distribution. The trace set is composed of system logs of WiFi access points on Dartmouth campus from September 1st, 2005 to October 4th, 2006. The number of sessions in every minute is extracted from the trace to generate the probability functions using rank regression.

5.2 Evaluation Results and Analysis

5.2.1 Machine Grouping Algorithm Performance Evaluation. In the first set of experiments, the accuracy of the proposed machine classification algorithm with multiple machine instances is evaluated. In order to improve the accuracy of the decision tree model proposed, the training set of machine instances should cover as many types of machine instances as possible. Since MIPS is used to represent the processing speed of machine (reflecting on CPU, cache and memory) that is either mobile device, or remote servers, we adopted the MIPS profiles obtained from CPU benchmarks [23], which include CPUs for desktops, laptops, and mobile devices. The overall available time of a machine is randomly generated from a uniform distribution. The network of machines in the mobile cloud system is generated by BRITE with different bandwidth on different links to represent multiple types of wireless medium. For the validation sets, three sets of machine instances are extracted from the real-world traces. The first set is a synthetic set that is generated from the same data distribution as the training set. The second and third validation set are extracted from the CRAWDAD trace.

Three features of the machines are considered in the machine grouping algorithm when classifying machines: processing speed, availability, and communication capacity. Processing speed is split into three sub-groups: fast, medium, and low. The availability and communication capacity are both split into two sub-groups: high and low. The weight factors $\alpha_{avail}, \beta_{avail}, \alpha_{conn}, \beta_{conn}$ in Equation 6 and 7 are all set to 0.5. Note that these weight factors will not influence the performance of the classification since both training sets and validation sets are using the same weight factor configuration to quantify machine features. The results of cross validation using WEKA are reported in Table 2.

As we can observe from the WEKA results, the grouping algorithm gives an accuracy of 99.5% on the synthetic machine instances and for the three real-world trace machine instances, it gives 99.79%, 99.65% and 99.8% accuracy respectively. Also, the reported accuracy remains similar while the number of instances grows. These

Table 3: Weight factor setting cases

Case	α	β	γ	Case	α	β	γ
1	1	0	0	7	0.2	0.6	0.2
2	0.8	0.1	0.1	8	0.2	0.2	0.6
3	0.6	0.3	0.1	9	0.1	0.7	0.2
4	0.6	0.1	0.3	10	0.1	0.2	0.7
5	0.4	0.4	0.2	11	0	1	0
6	0.4	0.2	0.4	12	0	0	1

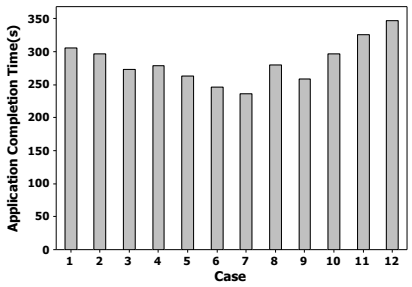
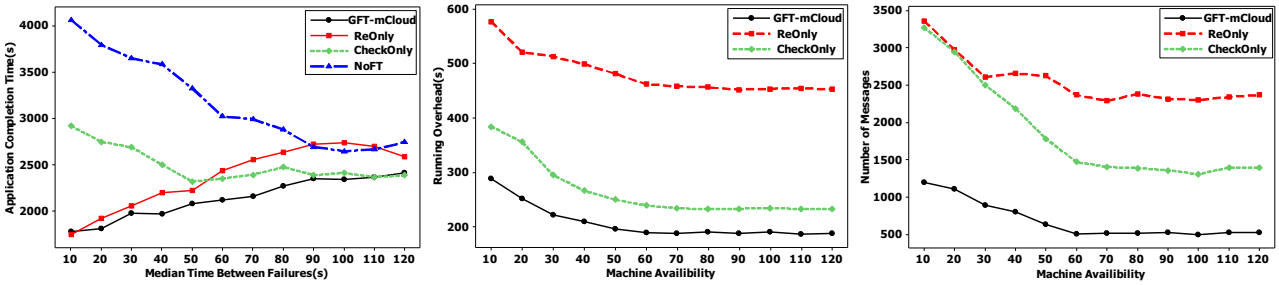


Figure 4: Average application completion time for different weight factor cases



(a): Overall application completion time

(b): Running overhead

(c): Number of messages for fault tolerance management

Figure 5: Performance under different machine availability

results show that the proposed machine grouping algorithm can accurately classify the machines into different groups based on the three features.

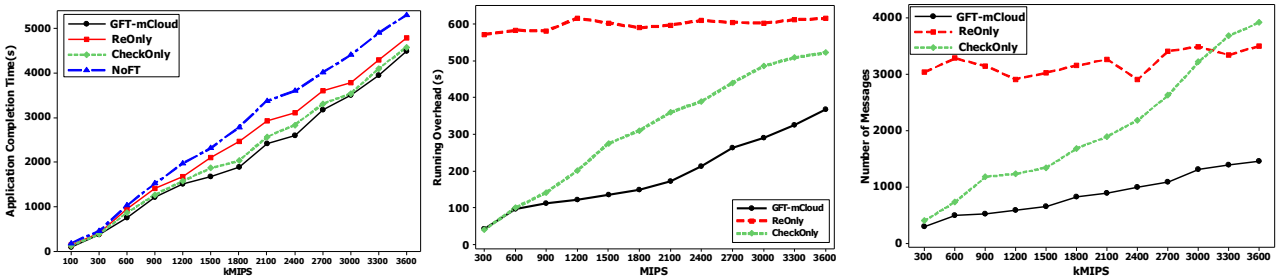
5.2.2 Group-based Fault Tolerant Mechanism Performance Evaluation. First, the weight factors α, β, γ used in the models (Equation 8) are set in different combinations to test their effects on the fault tolerance performance. The configurations of weight factors are listed in Table 3. The same network topology that contains a total number of 30 machines are used as the simulation environment. 50 randomly generated application DAGs are generated as the testing workload. The task offloading schedules of the application DAGs among the machine network used are obtained from the list scheduling heuristic HEFT (Heterogeneous Earliest Finish Time) [21] for the remaining sets of experiments in this section. The schedule plans are then put through the proposed FT mechanism and the other three policies for experiments. The results of application completion time (ACT) are depicted in Figure 4.

As the results show, the best weight factor setting with the lowest ACT is from case 7, which is $\alpha = 0.2, \beta = 0.6, \gamma = 0.2$. The average ACTs slightly decrease as the value of α decreases and the value of β increases. However, the ACT begins to increase again when the value of γ increases from 0.5 (case 6,8,10,12). These results indicate that the weight factor β that represents the weight of a machine's number of failures dominates the efficiency of the replication, and further influences the performance of GFT-mCloud in terms of overall application completion time. This is because when the mechanism selects replicas based on equation 8, the more reliable the machine is, the less probability that the replication fails to complete, which leads to a lower application completion time. The weight factor setting $\alpha = 0.2, \beta = 0.6, \gamma = 0.2$ will be used for the rest of the experiments in this section.

Secondly, we evaluate the GFT-mCloud in terms of the adaptivity on generating different fault tolerant policies based on different types of machines and tasks. The proposed mechanism is tested with 500 randomly generated application DAGs and machine networks, and results are averaged with 50 runs. Each DAG is generated with a random number of vertex as tasks and edges as task dependency. The tasks are coupled with data size and the amount of computation from the application profiling. The results are then compared with three other fault tolerance baselines: 1) replication only policy (**ReOnly**), 2) checkpointing only policy (**CheckOnly**), and 3) no fault tolerant policy (**NoFT**). **ReOnly** only applies replication of tasks as fault tolerance solution, while **CheckOnly** only uses checkpoint approach to maintain the task completion reliability. Three test cases are conducted below to evaluation the fault tolerance performance of GFT-mCloud. **Case A** study the influence of machine availability reflecting on its median time between failures. **Case B** and **case C** consider the effect of task computation and data size respectively on fault tolerance performance of the GFT-mCloud, since these two factors may affect the mobile cloud offloading.

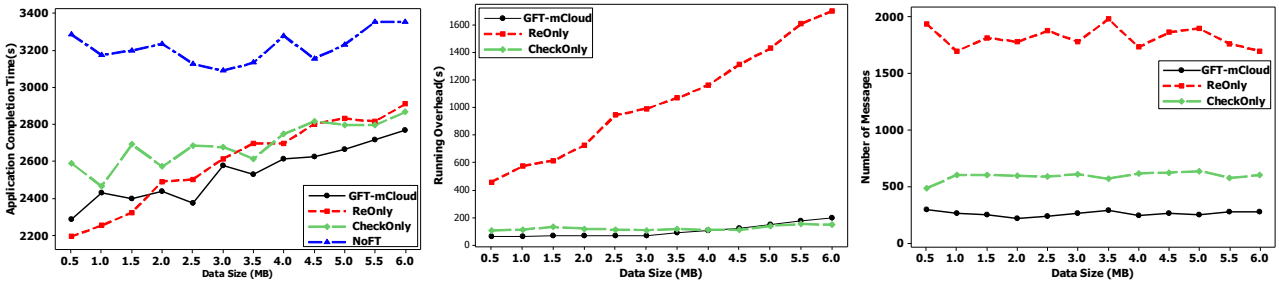
Test case A. In the first case, the performance of the proposed FT mechanism for different machine availability is evaluated. The machine availability is represented by the median of time between failures (MTBF) of the machines. The failure time points of each machine are generated from the Weibull distribution above-mentioned. The average application completion time of different machine availability are shown in Figure 5(a). As we can observe, GFT-mCloud outperformed the other three baselines. When the median time between failures is as small as 10 and 20 (i.e., machines are highly unavailable), the NoFT baseline generates the worst completion time of more 4000 seconds. CleckOnly generates the second worst completion time of around 3000. On the contrary, GFT-mCloud and ReOnly generated the application completion time below 2000, which is the lowest among the four algorithms compared. This is due to the fact that when the failure happens more frequent, the NoFT baseline and CheckOnly baseline keeps restarting the task execution, which leads to a higher overall application completion time than proposed mechanism and ReOnly baseline.

Moreover, when the machines have frequent failures, GFT-mCloud tends to apply replication as the policy, which makes its results similar to the ReOnly baseline when the MTBF is low. As the MTBF grows, the application completion time of NoFT and CheckOnly baseline decreases, and GFT-mCLoud and ReOnly baseline increases. This is because as the machine can execute tasks for longer time, the number of restarts for NoFT and CheckOnly becomes less as well as the remaining execution of the task if failures happen, which leads to a shorter makespan. Also, as machines become more available, the replicas are delayed to run on the backup machine since it needs to finish its own tasks first, which leads to a longer makespan for ReOnly. All four algorithms generate stable results when the median time between failures are greater than



(a): Overall application completion time (b): Running overhead (c): Number of messages for fault tolerance management

Figure 6: Performance under different task computation requirements



(a): Overall application completion time (b): Running overhead (c): Number of messages for fault tolerance management

Figure 7: Performance under different task data sizes

90. This is because as the machines become more reliable, ReOnly baseline generates more redundancy overhead for replica transmissions than CheckOnly baseline (Figure 5(b)). As the median time becomes longer, most of the tasks complete before failures happen, which makes the completion time generated more stable.

As shown in Figure 5(b) and Figure 5(c), the running overhead and number of messages incurred with the overhead are decreasing and become stable, which is consistent with the overall application completion time. The running overhead of the proposed FT algorithm accounts for around 9% after being stable.

Test case B. In the second case, the algorithms are evaluated with DAGs that have different amount of task computation requirements in terms of MIPS, to show the performance of the proposed FT mechanism with different applications. Machines used in this experiment are generated with the median time between failures from 90s to 120s. The average application completion time generated by four algorithms is depicted in Figure 6. When the computation amount is low, the application completion time generated by the four algorithms are almost same. As the computation requirement grows, the completion time grows in the similar speed, with NoFT giving the worst results among the four algorithms and GFT-mCloud outperforming the other three baselines. The growths of the completion time from the four algorithms are similar. This is consistent with the results shown in Figure 5 when the machine's median time between failures is around 90s to 120s. Additionally, GFT-mCloud performs the best when the average task computation requirement is above 1500 kMIPS as we can observe in Figure 6.

Additionally, Figure 6(b) and Figure 6(c) show the running overhead and control messages incurred by the redundancy of the fault tolerance management for GFT-mCloud, ReOnly and CheckOnly baselines. For ReOnly baseline, the overhead and number of control messages have not fluctuated much under machine availability change, because ReOnly baseline generates replicas for all tasks in the application. However, the running overhead and control messages increased for GFT-mCloud and CheckOnly since more checkpointing snapshots were generated when the task execution is longer.

Test case C. In the third case, the performance of the proposed FT mechanism is tested with different amount of data (i.e., data size) to see its effect in case of task migrations due to failures. The median of task computation in the workload is 1800 kMIPS. Machines used in this experiment are generated with the median time between failures from 90 seconds to 120 seconds. Results from the proposed algorithm along with the three baselines are shown in Figure 7.

In Figure 7(a), the makespans generated by NoFT, which are around 3300, do not fluctuate too much when data size grows. For GFT-mCloud and CheckOnly baseline, the makespan increased slowly as the data size grows, while the makespan of ReOnly increased more rapidly. This is due to the growth of data size having a greater impact on replication as each task has to maintain its replica in the beginning, but only small impact on checkpointing as it only

requires task migration when failure happens. It can be further observed from Figure 7(b) and Figure 7(c) that the data size growth has little impact on GFT-mCloud and ReOnly as the running overhead remain stable, while the running overhead of ReOnly increased due to the replication requires more time for migration and processing. Moreover, the number of fault tolerant related control messages for the three algorithms do not change much with the data size growth. Comparing to the results in Figure 5(c) and Figure 6(c), the number of control messages will only be affected by the machine availability and task computation amount instead of data size.

Discussion. The results from four sets of experiments conducted showed that the proposed GFT-mCloud is able to adapt its policies to different execution conditions, such as different machine availability and different applications (computation requirement and data size), and can generate the lowest makespan among the algorithms compared with around 5-10% running overhead and a small number of control messages. The application computation intensity (Figure 6) has the most effect on the performance of the proposed FT mechanism, while the data size of applications have limited impact of around 8% (Figure 7) in terms of makespan.

6 CONCLUSIONS AND FUTURE WORK

Mobile cloud computing environment consists of mobile devices, cloudlets, and remote cloud services, connected via wireless networks. The mobility of mobile devices, as well as the intermittent wireless connections, makes mobile cloud computing systems vulnerable to failures. As the existing fault tolerant algorithms for distributed systems only focus on machine crash failures, they do not suit mobile cloud environment that involves wireless network failures. In this paper, we proposed a group-based fault tolerance mechanism that works as a standalone module with the existing mobile cloud offloading frameworks. It classifies the machines in the mobile cloud network into different groups based on its capabilities using decision tree method. Different fault tolerant policies such as checkpointing and replication are adaptively applied based on the offloading tasks and the group of its offloaded machine in order to select a machine within that group for redundancy. The experimental results have shown that the proposed fault tolerance mechanism is able to adapt different machine capability and generate stable makespans, with low running overhead.

For the future work, we plan to investigate the optimal solution of checkpointing and replication frequency to achieve minimum energy consumption with the same level of performance on application makespan. We also want to investigate the effect of hard deadline applications and SLA violations of mobile cloud services on the performance of the proposed algorithms.

ACKNOWLEDGMENTS

The authors would like to thank Satish Narayana Srirama for providing advices on this work.

REFERENCES

[1] Sergio Barbarossa, Stefania Sardellitti, and Paolo Di Lorenzo. 2013. Joint allocation of computation and communication resources in multiuser mobile cloud computing. In *Proceedings of IEEE 14th Workshop on Signal Processing Advances in Wireless Communications*. IEEE, 26–30.

[2] Chien-An Chen, Myounggyu Won, Radu Stoleru, and Geoffrey G. Xie. 2015. Energy-efficient fault-tolerant data storage and processing in mobile cloud. *IEEE Transactions on Cloud Computing* 3, 1 (Jan 2015), 28–41.

[3] SungJin Choi, MaengSoon Baik, JoonMin Gil, SoonYoung Jung, and ChongSun Hwang. 2006. Adaptive group scheduling mechanism using mobile agents in peer-to-peer grid computing environment. *Applied Intelligence* 25, 2 (2006), 199–221.

[4] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI:: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. ACM, New York, NY, USA, 14.

[5] Menno Dobber, Rob van der Mei, and Ger Koole. 2009. Dynamic load balancing and job replication in a global-scale grid environment: A comparison. *IEEE Transactions on Parallel and Distributed Systems* 20, 2 (Feb 2009), 207–218.

[6] Richard O Duda, Peter E Hart, David G Stork, and others. 1973. *Pattern classification*. Vol. 2. Wiley New York.

[7] Jr. James E. Kelley. 1961. Critical-path planning and scheduling: Mathematical basis. *Operations Research* 9, 3 (1961), 296–320. arXiv:http://dx.doi.org/10.1287/opre.9.3.296

[8] Wolfgang Kiess and Martin Mauve. 2007. A survey on real-world implementations of mobile ad-hoc networks. *Ad Hoc Networks* 5, 3 (2007), 324 – 339.

[9] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. 2012. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings of the 31st IEEE International Conference on Computer Communications*.

[10] David Kotz, Tristan Henderson, Ilya Abyzov, and Jihwang Yeo. 2009. CRAWDAD dataset dartmouth/campus (v. 2009-09-09). Downloaded from http://crawdad.org/dartmouth/campus/20090909/syslog. (Sept. 2009).

[11] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (July 1982), 382–401.

[12] Antonios Litke, Dimitrios Skoutas, Konstantinos Tserpes, and Theodora Varvarigou. 2007. Efficient task replication and management for adaptive fault tolerance in Mobile Grid environments. *Future Generation Computer Systems* 23, 2 (2007), 163 – 178.

[13] James MacQueen and others. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA., 281–297.

[14] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. 2001. BRITE: An approach to universal topology generation. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '01)*. IEEE Computer Society, Washington, DC, USA, 346–.

[15] Nichamon Naksinehaboon, Yudan Liu, Chokchai Leangsuksun, Raja Nassar, Mihaela Paun, and Stephen L. Scott. 2008. Reliability-aware approach: An incremental checkpoint/restart model in hpc environments. In *Proceedings of 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*. 783–788.

[16] JiSu Park, HeonChang Yu, Hyongsoon Kim, and Eunyoung Lee. 2016. Dynamic group-based fault tolerance technique for reliable resource management in mobile cloud computing. *Concurrency and Computation: Practice and Experience* 28, 10 (2016), 2756–2769.

[17] J.R. Quinlan. 1986. Induction of Decision Trees. *Machine Learning* 1, 1 (1986), 81–106.

[18] M Reza Rahimi, Nalini Venkatasubramanian, and Athanasios V Vasilakos. 2013. MuSIC: Mobility-aware optimal service allocation in mobile cloud computing. In *Proceedings of the 6th IEEE International Conference on Cloud Computing*. IEEE, 75–82.

[19] Horst Rinne. 2008. *The Weibull distribution: a handbook*. CRC Press.

[20] Mahadev Satyanarayanan, P. Bahl, R Caceres, and N. Davies. 2009. The case for VM-based cloudlets in mobile computing. *Pervasive Computing, IEEE* 8, 4 (Oct 2009), 14–23.

[21] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13, 3 (Mar 2002), 260–274.

[22] Michael Treaster. 2005. A survey of fault-tolerance and fault-recovery techniques in parallel systems. *arXiv.org* abs/cs/0501002 (2005).

[23] Wikipedia. 2017. *Instructions per Second*. Technical Report.

[24] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[25] Liayo Xiang, Shiwen Ye, Yuan Feng, Baochun Li, and Bo Li. 2014. Ready, Set, Go: Coalesced offloading from mobile devices to the cloud. In *Proceedings of 2014 IEEE Conference on Computer Communications*. 2373–2381.

[26] John W. Young. 1974. A first order approximation to the optimum checkpoint interval. *ACM Communications* 17, 9 (Sept. 1974), 530–531.

[27] Bowen Zhou, Amir Vahid Dastjerdi, Rodrigo Calheiros, Satish Srirama, and Rajkumar Buyya. 2015. mCloud: A context-aware offloading framework for heterogeneous mobile cloud. *IEEE Transactions on Services Computing* (2015).