Cluster Comput (2014) 17:911–926
DOI 10.1007/s10586-013-0286-3

Fault tolerance and QoS scheduling using CAN in mobile social cloud computing

SookKyong Choi · KwangSik Chung · Heonchang Yu

Received: 16 February 2013 / Accepted: 2 June 2013 / Published online: 22 June 2013

Abstract The performance of mobile devices including smart phones and laptops is steadily rising as prices plummet sharply. So, mobile devices are changing from being a mere interface for requesting services to becoming computing resources for providing and sharing services due to immeasurably improved performance.

With the increasing number of mobile device users, the utilization rate of SNS (Social Networking Service) is also soaring. Applying SNS to the existing computing environment enables members of social network to share computing services without further authentication.

To use mobile device as a computing resource, temporary network disconnection caused by user mobility and various HW/SW faults causing service disruption should be considered. Also these issues must be resolved to support mobile users and to provide user requirements for services.

Accordingly, we propose fault tolerance and QoS (Quality of Services) scheduling using CAN (Content Addressable Network) in Mobile Social Cloud Computing (MSCC). MSCC is a computing environment that integrates social network-based cloud computing and mobile devices. In the computing environment, a mobile user can, through mobile devices, become a member of a social network through real world relationships. Essentially, members of a social network share cloud service or data with other members without further authentication by using their mobile device. We use CAN as the underlying MSCC to logically manage the locations of mobile devices. Fault tolerance and QoS scheduling consists of four sub-scheduling algorithms: malicious-user filtering, cloud service delivery, QoS provisioning, and replication and load-balancing. Under the proposed scheduling, a mobile device is used as a resource for providing cloud services, faults caused from user mobility or other reasons are tolerated and user requirements for QoS are considered.

We simulate scheduling both with and without CAN. The simulation results show that our proposed scheduling algorithm enhances cloud service execution time, finish time and reliability and reduces the cloud service error rate.

Keywords Cloud · Social cloud · Mobile social cloud · Fault tolerance · QoS · CAN

1 Introduction

Cloud computing is a model for enabling ubiquitous, convenient and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1].

Cloud computing is spreading to other computing areas such as mobile and social networks. Mobile cloud computing is a computing environment that extends cloud computing to include mobile devices such as laptops, netbooks, tablets, PDAs, UMPCs, smartphones, etc.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012R1A2A2A02046684).

S. Choi · H. Yu (✉)
Dept. of Computer Science Education, Korea University, Seoul, Korea
e-mail: yuhc@korea.ac.kr

S. Choi
e-mail: csukyong@korea.ac.kr

K. Chung
Dept. of Computer Science, Korea National Open University, Seoul, Korea
e-mail: kchung0825@knou.ac.kr

In mobile cloud computing, each mobile device should have a suitable application to access mobile cloud computing. Each mobile device can act as a server to provide cloud service or data as well as a client to request cloud service or data. In this case, these mobile devices are used as a resource in mobile cloud environments. However, given their nature, mobile devices can move freely in the network, which may cause network disconnection. Moreover, they may run out of power, resulting in a suddenly disconnection from the network and subsequent service suspension or cessation. Therefore, if mobile devices are to displace computing servers, these inherent problematic aspects of the mobile cloud must be overcome.

A social network is a social structure composed of individuals (or organizations) called "nodes", which are connected by one or more specific type of interdependency, such as friendship [2]. Essentially, a social network is a dynamic virtual organization with inherent trust relationships among friends.

Social Computing is a term for the intersection of social behavior and computational systems. In Social Computing, relations among individual users based on real world relationships create a digital relationship in the form of an online social network, and these relationships within online social networks have a fundamental level of implicit trust for sharing data and information. Therefore in social computing environments, users, through a social network, can share data and information among individual users based on real world relationships with low or even no authentication because members are willing to provide their mobile devices and their data to other social network members.

Mobile Social Cloud is a new paradigm. Tony Velleca [3] says that the convergence of cloud computing, mobile applications and social networking provides a powerful new paradigm for achieving disruptive innovation. Jacques Pavlenyi [4] also posits that the fusion of mobile devices, social networks and cloud computing is a new paradigm generating synergistic effects for the following reasons.

- *The ascendency of mobile computing*
  Mobility is good by itself. Mobile delivery of enterprise email, calendar and other critical applications has become a basic necessity.
- *The growth of social networks for business*
  Social collaboration like blogs, wikis, file sharing, and social document collaboration create great opportunities for productivity.
- *The continued rise of cloud computing*
  The cloud offers the promise of faster development and delivery of services while providing cost savings and faster iteration of new delivery services. The real beauty of the cloud is that users can seamlessly deliver services to multiple end-points such as tablets and PCs.

For the above reasons, we focus on Mobile Social Cloud Computing (MSCC), which integrates mobile devices, cloud computing, and social networking. In MSCC, a mobile device user requests cloud service from a cloud server and the user is informed of the closest mobile device of a user who belongs to the same social network and able to provide cloud service. Eventually the user requesting cloud service and the user of the closest mobile device are connected and share cloud service through the social network without further authentication. Using the closest mobile device for cloud service, not a faraway cloud server, improves cloud service execution time and reduces cloud service waiting time.

However, there are some issues in MSCC that must be dealt with due to the nature of mobile devices. MSCC has characteristics that distinguish it from traditional Grid computing as follows:

- *Network disconnection resulting from user mobility*
  MCSS supports user mobility. Therefore, users can freely move in a network, which may cause network disconnection from the AP or communication disconnection from the cloud server. These hinder user access to cloud services.
- *Inherent problem of mobile device*
  A mobile device is independently operated, so, the device may be turned off intentionally by the owner. Frequently, batteries run down. Moreover, there are some situations in which users cannot use cloud services because of mobile device faults, including physical defects, or software faults.

Given the characteristics of MSCC mentioned above, methods for tolerating faults and providing cloud services are essential. Accordingly, we propose a fault tolerance and QoS scheduling algorithm for mobile users to provide cloud services in the computing environment.

The contributions of this paper are as follows:

1. We propose a scheduling algorithm to support mobile users and share cloud services directly among users. To present MSCC environments, we use CAN (content-addressable network). CAN is a distributed network structure that has been used to manage moving objects and will be introduced in Sect. 3.
2. We propose a scheduling algorithm to tolerate faults in MSCC. Faults may bring loss of results for mobile user requests. In this paper, we regard all situations, including network disconnection, battery drain, and other failures due to mobile device movement as faults.
3. Additionally, we propose a scheduling algorithm to provide QoS (Quality of Services) to users in MSCC. We first classify QoS metrics for the Mobile Social Cloud and then apply the QoS metrics to MSCC.

Our scheduling algorithm is unique because the scheduling algorithm supports user mobility and user QoS concur-

rently while considering mobile device characteristics, and also supports cloud sharing services among mobile users without any authentication by using social network.

The rest of this paper is organized as follows. Section 2 introduces works related to this paper and discusses various existing methods to tolerate fault and support QoS for users in many computing environments. Section 3 describes the system environment for this paper. The section examines CAN, system architecture and data structure for MSCC. Section 4 proposes fault tolerance and QoS scheduling algorithm for MSCC consisting of four sub-algorithms: (1) malicious user filtering, (2) cloud service delivery, (3) QoS provisioning and (4) replication and load-balancing. Section 5 details the simulation results in MSCC and presents results with and without CAN. Section 6 summarizes the results and discusses various avenues for future research.

2 Related works

2.1 Fault tolerance in computing environment

Studies on methods to tolerate various faults that can occur during a service have been conducted in diverse research areas.

Jing Deng et al. [5] proposed matrix multiplication as a cloud selection strategy and technique to improve fault-tolerance and reliability and prevent faulty and malicious clouds in cloud computing environment having multiple clouds.

Jie Li et al. [6] proposed MODISAzure and used additional redundancy and fault-tolerance capabilities through retrying task execution, which supports debugging of user code encountering unanticipated data issues.

Yilei Zhang et al. [7] presented BFTCloud (Byzantine Fault Tolerant Cloud) for building robust systems in voluntary-resource cloud environments. They used replication techniques because computing resources in voluntary-resource clouds are heterogeneous and less reliable and malicious behaviors of resource providers cannot be prevented. They proposed a BFT group of one primary and 3f replicas for tolerating different types of failures. The primary and replicas form a BFT group for executing requests from the cloud task. If some nodes of the BFT group are identified as faulty, the cloud module will update the BFT group to guarantee system reliability.

Yi Hu et al. [8] proposed a security-aware and fault-tolerant job scheduling strategy for grid computing. The scheduling strategy includes JRT (Job retry), JMG (Job migration without checkpointing), and JCP (Job migration with checkpointing). They concluded that JRT strategy has the most optimal system performance improvement for small jobs and JCP strategy leads to the lowest performance improvement. Unfortunately, the checkpointing method was not described clearly.

Hyunjoo Kim et al. [9] proposed server selection schemes for a service migration-based fault-tolerant streaming on P2P computing.

2.2 QoS in computing environment

Various classification methods for QoS and QoS provisioning methods have been studied to meet diverse user needs for services. QoS is defined in different ways depending on the research area.

Qian Tao et al. [10] defined a Cloud_Services_QoS including Basic_ QoS Set and Extended_QoS Set and proposed a T_QoS (trustworthy QoS) computing algorithm. Basic_ QoS_Set considers time and cost, and Extended_QoS_Set includes reliability, availability, security, and reputation.

Sheikh Mahbub Habib et al. [11] provided a landscape of trust and reputation based approaches for selecting service providers in a cloud environment. They identified QoS+ (beyond QoS) parameters for cloud computing environments. In addition, they defined further parameters for performance tests (latency, bandwidth, availability, reliability and elasticity for performance tests, and crypto algorithms) and security measures (key management, physical security support, network security support, and data security support).

Meng Xu et al. [12] proposed a multiple QoS constrained scheduling strategy of multi-workflows, not a single QoS parameter such as execution time or cost. They used covariance for time and cost in cloud computing and tried to satisfy user QoS requirements by first scheduling a task with minimum covariance nodes.

Peng Zhang and Zheng Yan [13] proposed a QoS aware system for mobile cloud services that monitors the status of QoS in each node for mobile cloud services at run time. They used QoS properties for mobile devices like percentage of memory and CPU consumption, connection speed, remaining battery percentage and packet loss rate, etc.

Yanchao Zhang and Yuguang Fang [14] proposed a reputation system to predict the reliability of candidate servers for clients and support reliable service selection in P2P networks. They defined the reputation of a server as the probability that the server is expected to demonstrate a certain behavior, which is assessed by a client based on self-experiences with and other user feedback on the server. User QoS experiences are recorded on a data structure called a QoS experience vector. The reputation system uses reputation scores from QoS self-experience and support functions for fault tolerance and load balancing.

2.3 Service (data sharing) in cloud or social network environment

Many studies on service or data sharing among users and efficiency enhancement of the sharing have been conducted on cloud or social networks.

Juan M. Tirado et al. [15] proposed a data grouping and placement strategy on LastFM, an on-line music portal with social networking capabilities using cloud-based elastic server infrastructure. In the network, users have the possibility of mutually connecting through friendship.

Kyle Chard et al. [16] defined a social cloud as a resource and service sharing framework utilizing relationships established between members of a social network. The social cloud allows users to share heterogeneous resources with low privacy concerns and security overheads by utilizing the relationships in the computing environment.

Ryan Wooten et al. [17] proposed healthcare as a promising application of cloud computing and social media. They described the design and prototype of a social healthcare network through cloud computing and designed a trust-aware role-based access control to ensure the privacy and confidentiality of users.

2.4 Using CAN in computing environment

Most researches regarding CAN [18–21] focus on its utilization in P2P networks. There are only a few studies on CAN applicable to computing environments [22, 23].

Anuchart and Guang [24] proposed a new framework for the self-organizing and self-healing certificate authority (CA) group in CAN that provides certificates without a centralized trusted third party in P2P networks. To realize self-organizing and self-healing functionalities, the CA group itself maintains its CAN structure in a dynamic fashion by predefined group management policies and eliminates malicious nodes from the CA group through a Byzantine Agreement (BA) protocol. The multicast and broadcast communication is controlled by a CAN-based flooding algorithm. As P2P networks are composed of dynamic nodes that join, leave and fail unpredictably, they have characteristics similar to MSCC environments.

Henry et al. [25] introduced enhancement strategies to CAN for computational grids that provide transparent data distribution and replication. For enhancement to CAN, they chose CAN as basis for distributed storage algorithm because of its enormous flexibility. They also used an algorithm for supporting multiple data queries using CAN, an algorithm supporting locality optimization by reducing latency between underlying network structures using CAN routing, and an algorithm supporting multiple zones per node for flexibility. They improved CAN as an algorithmic basis for a distributed storage system.

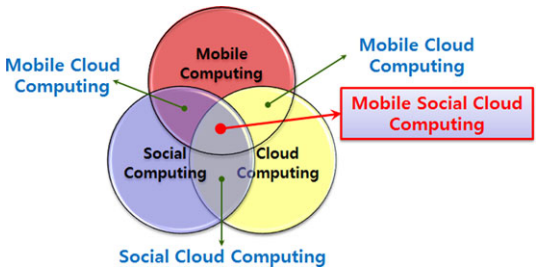

Fig. 1 Global view of MSCC

Roger et al. [26] proposed an approach to support spatial data management over structured P2P systems by extending CAN virtual space to physical spatial space and introduced a new hash function for mapping spatial data objects onto nodes over a modified CAN system. They identified the key of modified CAN for each data object as a bit string. However, they did not actually prove it.

3 System environment

3.1 Mobile Social Cloud Computing (MSCC)

MSCC refers to both (1) social networks and mobile cloud computing and (2) social cloud computing and mobile devices. We regard MSCC as the latter. Namely, we consider MSCC as social network-based cloud computing supporting mobile devices. We define MSCC as follows:

Definition 1 Mobile Social Cloud Computing (MSCC) is a computing model that includes mobile devices to support user mobility and connects with social networks to reflect real world user relationships, and therefore provides and shares cloud services directly among the members of a social network.

We assume MSCC, the basis of this paper, as follows:

1. MSCC consists of a number of mobile devices for requesting cloud services and computing resources for providing cloud services. Computing resources include some cloud servers and mobile devices.
2. Mobile devices in MSCC create social networks based on real world human relationships among mobile users. Thus a mobile user's device may belong to several social networks.
3. Members of a social network share cloud services based on basic authentication of the social network without any further authentication.

Figure 1 presents a global view of MSCC in various computing environments.

Figure 2 depicts cloud service utilization in MSCC. User1 and user2 are on the same social network and user2

Fig. 2 Cloud service utilization in MSCC

has cloud service or data. When user1 requests cloud service to a cloud server, the cloud server returns user2's mobile device information. Ultimately, user1 and user2 are connected and share the cloud service. Therefore, user1 does not have to connect to a distant cloud server.

3.2 CAN (Content Addressable Network)

Content Addressable Network (CAN) [18, 19] is a space partitioning mechanism that can be easily adapted for spatial data. Because CAN is a distributed infrastructure that provides hash table-like functionality, CAN has been used as a base approach for large scale data management of frequently moving objects in various computing environments: distributed hash table (DHT) and key-based routing protocol, etc. [20, 21].

Each CAN node uses the CAN routing mechanism to forward request messages, like insert, delete, and lookup, for a key. The message is then routed by intermediate CAN nodes towards the zone whose CAN node contains that key. For CAN routing, a routing table containing the IP addresses of the nodes is used. A node checks on which neighboring zone is closest to the destination point and looks up the IP address of a node in the closest neighboring zone via the routing table.

In this paper, we use CAN structure not to improve the performance of computing environment but rather to represent and manage network space consisting of mobile devices, as well as cloud servers, for the following reasons.

1. As CAN represents distributed, not centralized, infrastructure, it is suitable for mobile social cloud environments consisting of multiple distributed mobile devices and servers discussed in this paper.
2. CAN is able to represent a computing environment in which mobile devices join and leave networks freely. A key in CAN may be a mobile device, and CAN operations like insert and delete can be matched with joining and leaving of mobile devices in MSCC.
3. CAN maintains a hash table using (key, value) pairs for lookup. These (key, value) pairs can be mapped (cloud service, mobile device) pairs to determine the proper node for providing and sharing cloud services in MSCC.

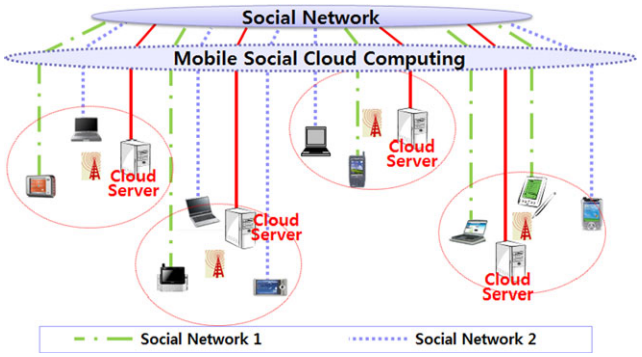

Fig. 3 Architecture of MSCC

4. CAN is used in peer-to-peer file sharing systems, such as Napster, in which data is stored in end-user devices rather than on a central server. The system is consistent with social network environments that share multimedia data with friends, family and others. Therefore, CAN is also used in MSCC.

3.3 Architecture of MSCC

MSCC, based on wired servers, includes mobile devices to support mobile users requesting cloud services in the network. Mobile users use mobile devices such as smartphones, laptops, PDAs, etc. that utilize mobile and wireless networks. These mobile devices can communicate with wired computers and other devices through AP (Access Point). There are multiple cloud servers in MSCC and each mobile device can access a cloud server. Cloud servers can provide services to several mobile devices. Every mobile device has a main cloud server that the mobile device is registered after joining the network. Mobile devices transmit their information to the main cloud server periodically.

Mobile users can be a member of a social network by using real world relationships and the members of a social network can share cloud service or data without further authentication. Not only cloud servers but also mobile devices provide cloud services to other members of the social network, so mobile devices act as resources.

Cloud servers and mobile devices have their positions determined by GPS in the network and can recognize the others' position through a main cloud server. They know who is a member of their social network through information stored in the main cloud server. Therefore, a mobile device requests cloud services to the closest cloud server or mobile device that is also a member of social network. This should result in improved service response time as it allows the sharing of cloud services without any further authentication.

Figure 3 depicts the architecture of an MSCC environment. There are some cloud servers and mobile devices connecting to APs. Mobile devices form a social network with

Table 1 Information stored in a resource

```
{   resource_ID;
    { x_coordinate, y_coordinate };
    AP_ID;
    current_zone_ID;
    { neighboring_zone_IDs };
    social_network_ID;
    { cloud_service_IDs };   }
```

Table 2 Information stored in a cloud server

```
{  resource {
   |   resource_ID;
   |   reputation;
   |   resourceability;
   |   cost;
   |   social_network_ID;
   |   other information received from Mobile Device;
   };
    cloud_service {
   |   service_ID;
   |   time;
   |   frequency;
   |   { provider_IDs; };
   };
    social_network {
   |   { social_network_ID, { members; } };
   };   }
```

other devices and cloud servers, and a cloud server can be a member of every social network to provide cloud services.

Every cloud server has a CAN structure to manage mobile devices. Every mobile device is registered on CAN in the cloud server and is mapped on a point of CAN having a virtual logical address, namely CAN coordinates, for CAN routing.

3.4 Preliminary

In MSCC, a mobile device can be used as a resource, so the mobile device requests and provides cloud service using a resource ID. Resource ID is managed by the resource along with GPS-based coordinates (x, y), AP id, current and neighboring zone IDs of CAN, social network ID, and cloud service IDs. GPS-based coordinates are used to search for a resource in a neighboring cloud server and compute the distance between the resource and a requested resource. Zones of CAN are used to look up any resource on cloud servers. Cloud service ID refers to the ID of a cloud service that the resource can provide to other resources. This information is periodically sent to the main cloud server. Table 1 shows information stored in a resource.

Each cloud server stores and manages resource information, including resource ID, reputation, resourceability, cost, social network ID, etc. Reputation and resourceability are introduced in Sect. 4. Cloud servers also store cloud service information such as service ID, time, frequency, and IDs of resources that can provide the cloud service. Moreover, the cloud server stores social network information, including social network ID and list of members constructed from social network ID transmitted by mobile devices. The information stored in a cloud server is summarized in Table 2.

When a cloud server receives a request for cloud service, the server looks for the most proper resource to provide the service. For cloud service delivery, the algorithm to find the most proper resource is introduced in Sect. 4.

4 Fault tolerance and QoS scheduling

4.1 Filtering for malicious user

It is not fair if one user only requests a cloud service and another user only provides a cloud service in a network. For a sound and healthy network, there should not be a unilateral service or data flow. In this scheduling, the basis of usage of networks is mutual communication among users of the network.

Malicious in this paper includes all user acts that only use cloud services from other users or avoid providing cloud services to others. Malicious users may reject another's request when getting a request for cloud services from another user or disconnect another user forcibly even while providing cloud services to another user.

Because malicious users reduce cloud service quality, we exclude malicious users from MSCC network configurations. To do this, we use mobile device reputation. If someone had behaved badly or maliciously in sharing of cloud services previously, that person will have a low reputation.

We define the reputation of a mobile device as follows [14].

Definition 2 Reputation is the probability of a mobile device that can be assessed by a server or other mobile device on the basis of feedback. Reputation is used to determine whether a user of mobile device is malicious or not in a Mobile Social Cloud network.

4.1.1 Calculation of reputation

An MSCC network is represented by CAN. Mobile devices join the network by communicating with a randomly selected node in CAN and then managed by a main cloud server. The main cloud server manages the mobile device after the mobile device joins CAN.

The main cloud server sends a unique resource identifier (resource_ID) and test application to a newly joined mobile device. After some time, the main cloud server requests the result of the test application to the mobile device. Malicious users may not return the result, return an error result, or return part of result. Namely, the test application is used to

calculate the reputation value of a mobile device and determine whether the user is malicious or not.

After the mobile device transfers the result of the test application to the main cloud server, the main cloud server calculates the reputation value of the mobile device by comparing the original answer of the test application and the result received from the mobile device. If the reputation value of the mobile device is below a specified threshold, the main cloud server evaluates the mobile device as malicious.

We use a series of random matrices for the test applications. The initial reputation, $reputation_0$, of a mobile device can be attained as shown in Eq. (1) in which m is the number of matrix in a series and n is the number of matrix element in a matrix. The expression CS_{ij} represents a value of the answer matrix element for the test application in a main cloud server, and MD_{ij} represents a value of the result matrix element that is calculated in the mobile device and submitted to the main cloud server.

$$reputation_0 = \sum_{i=1}^{m}\sum_{j=1}^{n} RS_{ij}$$
$$RS_{ij} = \begin{bmatrix} 0 & (CS_{ij} \neq MD_{ij}) \\ 1 & (CS_{ij} = MD_{ij}) \end{bmatrix} \quad (1)$$

If the mobile device is reliable, the reputation value is high. Conversely, the lower the reputation value, the higher the probability of malicious user. If the initial reputation $reputation_0$ of a mobile device is below a certain threshold, the mobile device may not be connected to that particular MSCC. That means the mobile device can neither request nor provide cloud service in the network.

Because the status of a mobile device constantly changes, reputation is recalculated periodically by Eq. (2) to consider changes. The notation α is a weighted value for current reputation ranging from 0 to 1. Also, if the reputation $reputation_i$ of a mobile device is below the threshold, the mobile device can neither request nor provide cloud service.

$$reputation_{i+1} = \begin{bmatrix} reputation_i & (if\ i = 0) \\ reputation_i * \alpha + reputation_{i-1} * (1-\alpha) & (if\ i \geq 1) \end{bmatrix} \quad (2)$$

The average reputation of a mobile device for some duration is calculated by Eq. (3). N is the number of calculations in the duration. In Sect. 4.3, the average reputation is used for applying mobile QoS to select a proper resource.

$$avg\_reputation_{MD} = \sum_{i=0}^{N-1} reputation_i / N \quad (3)$$

Algorithm 1 Algorithm for network construction

```
network_construction() {
    send a unique resource_ID and test application to newly
        joined mobile device;
    for (a time unit);
    request result of the test application to the mobile device;
    if (receive result from the mobile device) {
        compare the correct answer with the received result;
        compute an initial reputation of the mobile device
            using Eq. (1);
    }
    else
        set the reputation of the mobile device to minimum
            value;
    for (some time unit);
    recalculate the reputation of the mobile device using
        Eq. (2);
}
```

4.1.2 Algorithm for network construction

Through reputation, a mobile device that has just joined the network can be evaluated as to whether it may construct the network or not.

If the initial reputation is less than the threshold, the mobile device is excluded from the construction of the network for a particular time unit, which means the device can neither obtain nor provide cloud service. If the main cloud server does not receive a result from the mobile device, the main cloud server sets the reputation of the mobile device to the minimum value, labeling the mobile device as malicious. After some time unit, the main cloud server recalculates the reputation of the mobile device.

Algorithm 1 shows the algorithm for MSCC network construction in a main cloud server.

4.2 Cloud service delivery with social network

4.2.1 Basic algorithm for cloud service delivery

When a cloud server receives a request for cloud service from a mobile device, the server checks first whether the mobile device and the cloud server are in the same network area, namely the same AP area. If so, the cloud server directly provides cloud service to the mobile device. If not, the server searches for a proper resource that has the requested cloud service and is managed by the cloud server's CAN.

The cloud server puts searched resources into the first candidate group. Then, the cloud server checks the social network of the mobile device and selects resources belonging to the social network of the mobile device, that is, the same social network among the first candidate group. These selected resources are included in the second candidate group.

If there are no proper resources in the second candidate group, the server may then transfer the request to the closest neighboring cloud server with the resource_ID of the mobile device. CAN based-coordinates are used to search the proper resource in a cloud server, and GPS based-coordinates are used to identify the closest neighboring cloud server.

In sequence, the cloud server selects a proper resource by calculating the Euclidean distance between the two resources, mobile device A requesting cloud service and resource B included in the second candidate group, in Eq. (4). If the coordinates of mobile device A is $A(p_1, p_2)$, then the coordinate of resource B is $B(q_1, q_2)$.

$$distance_{AB} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{4}$$

After searching for the most appropriate resource, the cloud server returns its resource_ID to the mobile device requesting cloud service. When the requested cloud service is completed, the mobile device informs the cloud server.

Finally, the cloud server updates the reputation of the proper resource and the frequency of the requested cloud service. If the resource does not provide cloud service to the mobile device, the reputation of the resource will decline.

In addition, if the frequency of a cloud service is greater than the threshold, the cloud server replicates the cloud service to another neighboring cloud server according to the replication algorithm in Sect. 4.4.

Algorithm 2 shows the basic algorithm for cloud service delivery in a cloud server.

Algorithm 2 Algorithm for Cloud Service Delivery

```
service_delivery(service_ID) {
  if (receive a request for cloud service(service_ID)) {
    frequencyservice_ID = frequencyservice_ID + 1;
    check location of the mobile device requesting a
      cloud service;
    if (same AP area) {
      proper_resource_ID = server_ID;
    }
    else {
      proper_resource_ID =
        find_resource(mobile_device_ID, service_ID);
    }
    return proper_resource_ID;
  }
  else {
    for (a time unit);
    frequencyservice_ID = frequencyservice_ID – 1;
  }
  if (receive a cloud_service_finish_message from mobile
      device(resource_ID)) {
    compute reputationresource_ID;
  }
}
find_resource(mobile_device_ID, service_ID) {
  search resources managed by the cloud server and
    having service_ID;
  put resources into the first candidate group;
  for (each resource in the 1st candidate group) {
    check their social_network_ID
    if (same social network with the mobile device) {
      put searched resource into the second
      candidate group;
      for(each resource in the 2nd candidate group) {
        compute distance between mobile device
        and the resource using Eq. (4);
      }
    }
  }
  select one proper resource with minimum distance from
    second candidate group;
  return resource_ID of the proper resource;
}
```

4.3 QoS provisioning

4.3.1 QoS metrics for MSCC

We propose QoS metrics for MSCC to consider user needs and mobile device characteristics. We classify QoS metrics into common QoS and mobile QoS. Common QoS refers to general QoS for most users and includes time and cost. Time is task processing time for executing a task, and cost is task processing cost for using a cloud resource. Mobile QoS denotes specialized QoS on a mobile device and includes reputation and resourceability. Figure 4 shows QoS metrics for MSCC.

Reputation is already described in Sect. 4.1 and is subject to a mobile user's usage patterns based on historical information. We use average reputation by Eq. (3) for QoS. We define resourceability as follows.

Definition 3 Resourceability is the probability that a mobile device can be used to provide cloud service to other mobile devices, i.e., the ability to be a resource for a mobile device.

Resourceability is a metric that includes remaining battery power, network status, and performance of a mobile device. Resourceability prevents faults that can arise from the movement or other error of mobile devices during execution. That means mobile devices with low remaining battery power, poor network status, or low performance will not be selected as resources. Inversely, mobile devices with high remaining battery power, good network status, and high performance are eligible to provide cloud service to other mobile devices.

Users desire a resource with quick time, low cost, higher reputation, and better resourceability. Because a resource with higher reputation and resourceability generally has high cost, a cloud server will try to select a resource with

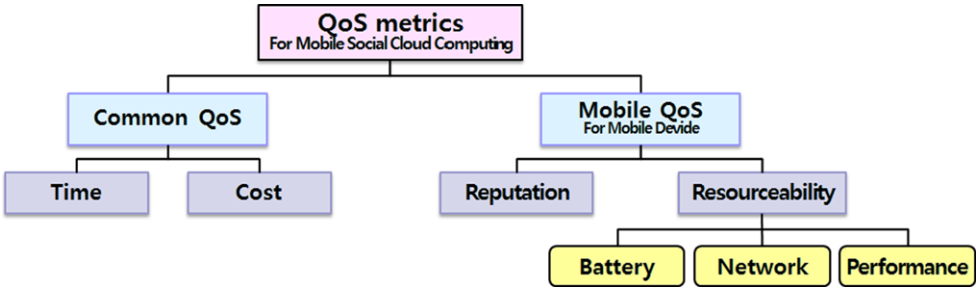

Fig. 4 QoS metrics for MSCC

low cost and higher reputation and resourceability for the user whenever possible.

4.3.2 Calculation of resourceability

Resourceability, used as a criterion to select the most proper mobile device, consists of battery power, network status and performance. These three variables are independent and affect resourceability; therefore, we use multiple regression analysis. The resourceability value, ranging from 0 to 1, of a mobile device is computed through Eq. (5). Variables x_1, x_2, and x_3 represent remaining battery power, network status, and performance of a mobile device, respectively. Parameters β_0, β_1, β_2 and β_3 mean the regression coefficients and ε means the error rate in multiple regression.

$$resourceability = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon \tag{5}$$

The higher the resourceability value, the higher the probability of the mobile device to be the most proper resource.

Because a mobile device with low remaining battery power or poor network status cannot be used as a resource, if the value of x_1 and x_2 is less than or equal to the threshold, we set the resourceability value of the mobile device to zero.

4.3.3 Algorithm for QoS provisioning

If a user wants to use QoS for cloud service, the user requests cloud service through the QoS metrics. Namely, a mobile device user requesting a cloud service can set the QoS metrics. If QoS metrics including common QoS and mobile QoS are set, the QoS algorithm is applied instead of the basic algorithm for cloud service delivery, as seen in Sect. 4.2.

When a cloud server receives QoS metrics from a mobile device, the cloud server applies the QoS metrics to MSCC. First, the cloud server checks whether the mobile device and the cloud server are in the same network area, as in Sect. 4.2. If so, the cloud server provides cloud service to the mobile device directly. The reason is that a cloud server is unaffected by remaining battery power, its network status and performance is good and a cloud server is included in every social network. If the two are not in the same network area, the server searches for resources, along the lines of Sect. 4.2.

After obtaining the second candidate group, the cloud server applies common QoS to the second candidate group and obtains resource group R_{tc} as a result. Namely, resource group R_{tc} is a set of resources satisfying time and cost among the second candidate group.

$$R_{tc} = \{mobile\ devices\ satisfying(QoS_{time} \times QoS_{cost})/ second\ candidate\ group\}$$

If resource group R_{tc} has only one resource, the cloud server returns the resource_ID of the resource to the mobile device. If resource group R_{tc} includes more than one resource, the cloud server applies mobile QoS to resource group R_{tc} and obtains resource group R_{rr} as a result. $QoS_{reputation}$ and $QoS_{resourceability}$ are calculated by Eqs. (3) and (5), respectively

$$R_{rr} = \{mobile\ devices\ satisfying \times (QoS_{reputation} \times QoS_{resourceability})/R_{tc}\}$$

If resource group R_{rr} has only one resource, it becomes the proper resource. If resource group R_{rr} includes more than one resource, the cloud server ranks the resources in group R_{rr} by calculating the distance between the mobile device requesting cloud service and a resource included in R_{rr} using Eq. (4). The cloud server selects the highest ranked resource as the most proper resource.

After selecting the appropriate resource, the cloud server returns its resource_ID to the mobile device requesting cloud service. When the requested cloud service is completed, the mobile device informs the cloud server to that effect.

Finally, the cloud server updates the reputation of the proper resource and the frequency of the requested cloud service, as in Sect. 4.2.

Algorithm 3 presents the QoS algorithm in a cloud server.

Algorithm 3 Algorithm for QoS

```
service_delivery(service_ID) {
  if (receive a request for cloud service(service_ID)) {
    frequency_service_ID = frequency_service_ID + 1;
    check location of the mobile device requesting a
        cloud service;
    if (same AP area) {
      proper_resource_ID = server_ID;
    }
    else {
      proper_resource_ID =
      find_resource(mobile_device_ID, service_ID);
    }
    return proper_resource_ID;
  }
  else {
    for (a time unit);
    frequency_service_ID = frequency_service_ID − 1;
  }
  if (receive a cloud_service_finish_message from
      mobile device(resource_ID)) {
    compute reputation_resource_ID;
  }
}
find_resource(mobile_device_ID, service_ID) {
  search resources managed by the cloud server and
      having service_ID;
  put resources into the 1st candidate group;
  for (each resource in the 1st candidate group) {
    check their social_network_ID;
    if (same social network with the mobile device) {
      put searched resource into the 2nd candidate
          group;
      apply common QoS to the 2nd candidate
          group;
      for (each resource satisfying common QoS) {
        put resources into resource group R_tc;
        apply mobile QoS to R_tc;
        for (each resource satisfying mobile QoS) {
          put resources into resource group R_rr;
          rank resources in group R_rr and select
              one proper resource;
          return resource_ID of the proper
              resource;
        }
      }
    }
  }
}
```

4.4 Replication and load-balancing

4.4.1 Algorithm for replication of service request

Mobile devices may have problems such as battery drain, software error and network disconnection. Therefore, given these faults, even after a mobile device receives a request for cloud service, the mobile device may not provide cloud service to other mobile devices. It is essential to deal with these faults.

Replication helps improve service availability, service completion rate and load balancing of the entire network because the request for cloud service can be distributed to replicas. Therefore, we use cloud service replication to minimize waiting time for the request and improve MSCC performance.

Algorithm 4 Algorithm for Replication in a Cloud Server

```
service_replication_request_in_CS(service_ID) {
  if (receive a request for cloud service(service_ID)) {
    frequency_service_ID = frequency_service_ID + 1;
    check location of the mobile device requesting a
        cloud service;
    if (same AP area) {
      proper_resource_ID = server_ID;
    }
    else {
      proper_resource_ID1 =
      find_resource(mobile_device_ID, service_ID);
      do {
        proper_resource_ID2 =
        find_resource(mobile_device_ID,
            service_ID);
      } while (proper_resource_ID1 =
          proper_resource_ID2);
    }
    return (proper_resource_ID1,
                                proper_resource_ID2);
  }
  else {
    for (a time unit);
    frequency_service_ID = frequency_service_ID − 1;
  }
  if (receive a cloud_service_finish_message from
      mobile device(resource_ID)) {
    compute reputation_resource_ID;
  }
}
```

When a cloud server receives a request for cloud service, the cloud server looks for two more proper resources and returns their resource_IDs using the cloud service delivery algorithm or QoS algorithm.

A mobile device requests the cloud service to the resources having the resource_IDs that have been received from the cloud server. If the mobile device cannot connect with one resource or some faults occur over the cloud service, the mobile device asks the other resource to provide cloud service.

Algorithms 4 and 5 show the algorithm for replication of service request in a cloud server and a mobile device, respectively.

4.4.2 Algorithm for replication with load-balancing

Cloud servers check the cloud service frequency periodically. Frequency increases whenever a request for the cloud service is issued and decreases whenever a time unit passes without any request for cloud service.

If the frequency of a cloud service exceeds the threshold, the cloud service is deemed to be popular and the cloud server replicates the cloud service to other neighboring cloud servers. For replication, a cloud server informs a neighboring cloud server to replicate a cloud service and transfers the cloud service.

Algorithm 5 Algorithm for Replication in a Mobile Device

```
service_replication_request_in_MD(service_ID) {
    request for cloud service(service_ID) to cloud
        server;
    if (receive 2 proper_resource_IDs from cloud server) {
        select one resource(proper_resource_ID1);
        connect with the resource;
        ask the resource to provide cloud
            service(service_ID);
        if (there is any fault over cloud service) {
            connect to another resource(proper_
                resource_ID2);
            ask the resource to provide cloud service
                (service_ID);
        }
    }
    if (finish cloud service(service_ID) {
        inform the cloud server of finishing of the service;
    }
}
```

When a neighboring cloud server receives the notification of replication, it stores the cloud server and set the frequency value of the cloud service to one.

After some time unit, if there is no request for a cloud service and the frequency is zero, a cloud server deletes the cloud service.

Algorithms 6 and 7 present the algorithm for replication supporting load balancing in a cloud server and the neighboring cloud server, respectively.

5 Performance evaluation

5.1 Experimental setup

We use CloudSim [27] to simulate an MSCC environment. CloudSim is a framework for modeling and simulation of cloud computing infrastructures and services.

In order to build a network that includes mobile devices, we use a network topology generator, BRITE (Boston University Representative Internet Topology gEnerator), a tool that generates realistic Internet topologies. Each BRITE result value corresponds to the geographical location of a mobile device.

We assume that there is no data transfer delay between mobile devices belonging to the same AP. In addition we assume that there is no usage error for cloud services between SNS members in the same SNS Group.

For the simulation, we set the configuration for simulation as Table 3.

We classify the simulation environments into four cases according to (1) the filtering of malicious users, (2) use of SNS, (3) use of user QoS, and (4) use of service replication. According to the algorithm for QoS Provisioning in Sect. 4.3, cases supporting user QoS but not supporting SNS concurrently among them are not considered in this paper. That is, we exclude these four cases for the simulation.

Algorithm 6 Algorithm for Replication with Load-balancing in a Cloud Server

```
replication_with_loadbalancing(service_ID) {
    if (receive a request for cloud service(service_ID)) {
        frequency_service_ID = frequency_service_ID + 1;
        check location of the mobile device requesting a
            cloud service;
        if (same AP area) {
            proper_resource_ID = server_ID;
        }
        else {
            proper_resource_ID1 =
            find_resource(mobile_device_ID, service_ID);
            do {
                proper_resource_ID2 =
                find_resource(mobile_device_ID, service_
                    ID);
            } while (proper_resource_ID1 = proper_
                resource_ID2);
        }
        return (proper_resource_ID1, proper_resource_
            ID2);
    }
    else {
        for (a time unit);
        frequency_service_ID = frequency_service_ID − 1;
    }
    if (receive a cloud_service_finish_message from
        mobile device(resource_ID)) {
        compute reputation_resource_ID;
    }
    if (frequency_service_ID >= threshold) {
        neighbor_server_ID =
            find_neighbor_server(server_ID, service_ID);
        replication(neighbor_server_ID, service_ID);
    }
    for (some time unit);
    if (no request for Service_ID and frequency_service_ID = 0)
        delete the cloud service(service_ID);
}
find_neighbor_server(server_ID, service_ID) {
    find the closest neighboring server from cloud
        server(server_ID);
    if (a neighboring server does not have the cloud
        service(service_ID)
        return the neighboring server ID;
    else find another neighboring server;
}
```

Algorithm 7 Algorithm for Replication with Load-balancing in Neighboring Server

```
replication_with_loadbalancing_in_neighbor_server(servic
e_ID) {
    if (receive replication message from a server(server_ID)) {
        store the cloud service(service_ID);
        frequency_service_ID = 1;
    }
    for (some time unit);
    if (no request for Service_ID and frequency_service_ID = 0)
        delete the cloud service(service_ID);
}
```

Table 4 shows the twelve cases for simulation according to fault tolerance and QoS scheduling. We simulated 30 times for each case.

5.2 Evaluation results and analysis

To evaluate the scheduling algorithm performance of 50 cloud services, we use the following criteria: (1) average execution time, (2) finish time, (3) reliability and (4) error rate of cloud services. We also use replication factor, i.e., the number of replicated service is set to 1, for the replication and load-balancing algorithm.

Table 3 Configurations for simulation

- number of mobile devices: 100
- number of VM(virtual machine)s: 30
- number of cloud services: 50
- number of AP(access point)s: 4
- number of SNS(social network service)s: 3

Table 4 Simulation cases

	Filtering of malicious users	SNS	User QoS	Service replication
Case1	No	No	No	No
Case2	No	No	No	Yes
Case3	No	Yes	No	No
Case4	No	Yes	No	Yes
Case5	No	Yes	Yes	No
Case6	No	Yes	Yes	Yes
Case7	Yes	No	No	No
Case8	Yes	No	No	Yes
Case9	Yes	Yes	No	No
Case10	Yes	Yes	No	Yes
Case11	Yes	Yes	Yes	No
Case12	Yes	Yes	Yes	Yes

CAN structure does not affect either fault tolerance or QoS scheduling performance and is only used to make a logical structure of the mobile devices.

Table 5 and Fig. 5 compare each case of evaluation criteria. Table 6 compares each case of evaluation criteria through ranking from 1 to 12.

With the CAN structure, for average execution time, case 12 is the fastest and case 2 is the slowest. For finish time, case 6 is the fastest. For reliability, case 10 is the most reliable. For error rate, case 12 is the most reliable. Without the CAN structure, for average execution time, case 12 is the fastest. For finish time, case 6 is the fastest. For reliability, case 11 is the most reliable. For error rate, case 12 is the most reliable.

From here on, because there is little difference between with and without CAN, we consider only cases of evaluation criteria with CAN.

5.2.1 Cloud service execution time

Under the same conditions, cases that filter malicious users and use SNS and user QoS are faster and those replicating cloud service are slower. It seems that filtering malicious users and using SNS and user QoS ensure reliable cloud service among SNS members because as there are fewer errors during cloud service, there is no need to re-provide cloud service and therefore average execution time decreases. Conversely, service replication is slower because extra overhead, such as selecting a resource and determining better cloud service results, is necessary for replication.

Table 7 shows the average execution time for each scheduling algorithm according to use or not option. Figure 6 shows the average execution time for each case.

Table 5 Comparison of evaluation results

CASE	Execution time		Finish time		Reliability		Error rate	
	without CAN	with CAN	without CAN	with CAN	without CAN	with CAN	without CAN	with CAN
1	602.0	627.5	2097.8	2175.8	0.55	0.52	0.48	0.45
2	873.5	837.1	1637.0	1580.5	0.63	0.64	0.36	0.37
3	506.5	498.5	2166.5	2066.9	0.74	0.76	0.24	0.26
4	612.2	581.8	1602.8	1675.2	0.74	0.73	0.27	0.26
5	357.3	353.3	1423.3	1480.5	0.78	0.77	0.23	0.21
6	402.7	404.7	1262.8	1332.2	0.75	0.76	0.20	0.21
7	536.5	517.6	2275.8	2114.8	0.76	0.74	0.26	0.24
8	586.0	646.4	1683.0	1775.3	0.73	0.74	0.26	0.27
9	508.5	492.4	2169.2	2121.6	0.92	0.93	0.06	0.07
10	407.6	415.7	1649.1	1642.8	0.94	0.91	0.08	0.06
11	341.8	344.9	1456.0	1501.6	0.92	0.97	0.00	0.00
12	313.7	327.8	1359.5	1340.0	0.93	0.92	0.00	0.00

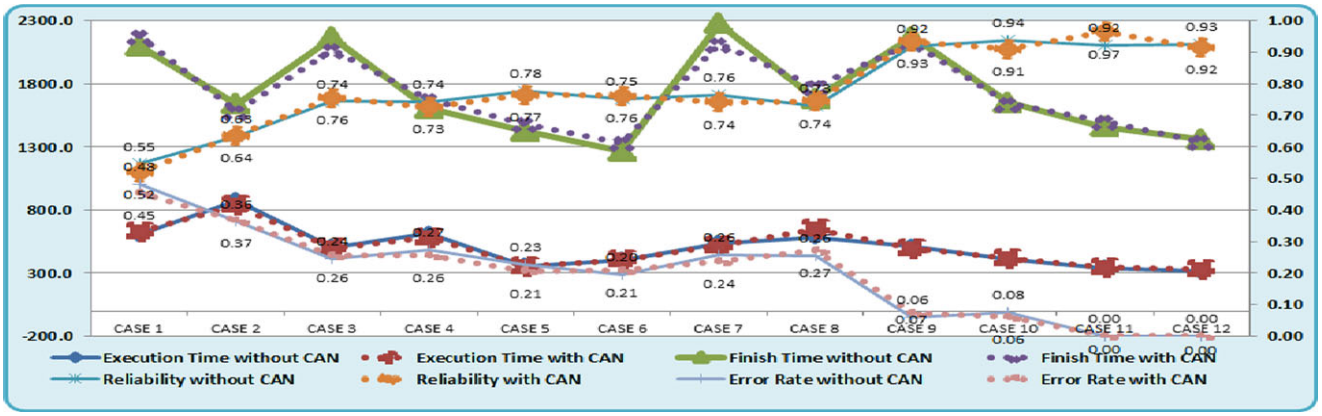

Fig. 5 Comparison of evaluation results

Table 6 Comparison of evaluation results in ranks

CASE	Execution time		Finish time		Reliability		Error rate	
	without CAN	with CAN	without CAN	with CAN	without CAN	with CAN	without CAN	with CAN
1	10	10	12	9	12	12	12	12
2	12	12	5	6	11	11	11	11
3	7	6	9	10	7	8	8	7
4	9	11	7	5	10	8	9	10
5	3	3	3	3	5	5	5	6
6	4	4	1	1	6	7	6	5
7	8	8	10	12	9	6	7	8
8	11	9	8	8	8	10	10	8
9	6	7	11	11	2	3	4	3
10	5	5	6	7	4	1	3	4
11	2	2	4	4	1	3	1	1
12	1	1	2	2	3	2	1	1

Table 7 Cloud service execution time with CAN

Use or not	Filtering of malicious users	SNS	User QoS	Service replication
No	550.48	657.16	577.13	**472.35**
Yes	**457.47**	**427.38**	**357.65**	535.59

Table 8 Cloud service finish time with CAN

Use or not	Filtering of malicious users	SNS	User QoS	Service replication
No	**1718.52**	1957.04	1894.12	1910.20
Yes	1749.35	**1645.10**	**1413.57**	**1557.67**

5.2.2 Cloud service finish time

Under the same conditions, cases that use SNS, user QoS, and service replication are faster than those not using them. Regarding the filtering of malicious users, there is no apparent difference between use and non-use. Using SNS and user QoS is faster as in Sect. 5.2.1. For service replication, using the algorithm is more optimal as, between the original and replication cloud services, one cloud service finished earlier than the other. Its execution time is set to the cloud service execution time.

Table 8 shows the average finish time for each algorithm according to the use or not option and Fig. 7 shows the average finish time for each case.

5.2.3 Cloud service reliability and error rate

Under the same conditions, all cases for each algorithm produce a more optimal result than when the algorithm is not used. For all cases, using each scheduling algorithm improves cloud service reliability and simultaneously reduces the cloud service error rate. For service replication, using service replication does not seem to have much effect on

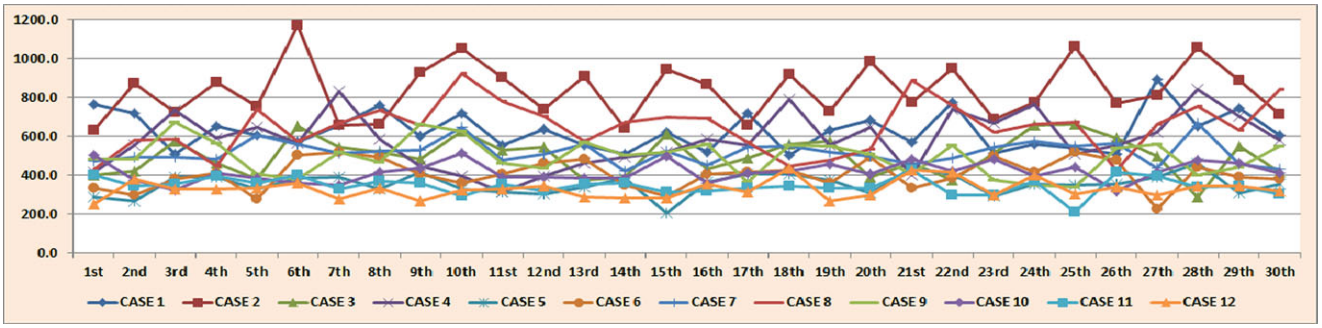

Fig. 6 Cloud service execution time with CAN

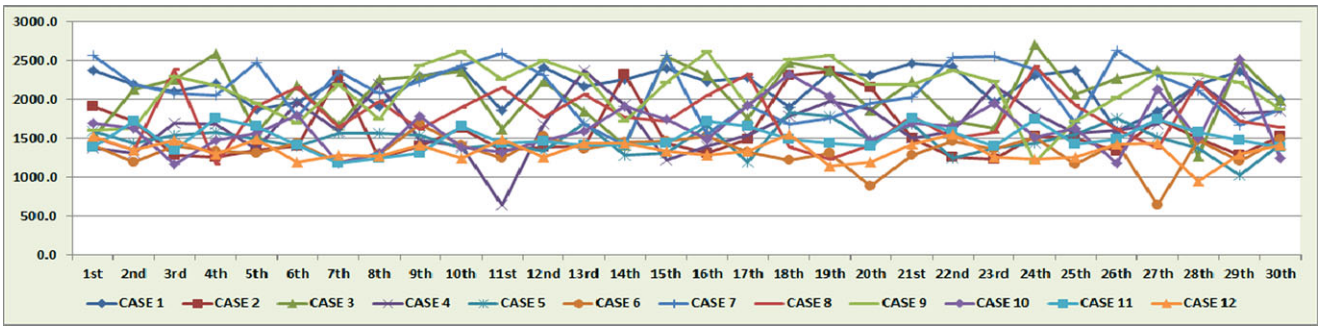

Fig. 7 Cloud service finish time with CAN

Table 9 Cloud service reliability with CAN

Use or not	Filtering of malicious users	SNS	User QoS	Service replication
No	0.69	0.66	0.75	0.78
Yes	**0.87**	**0.84**	**0.85**	**0.78**

Table 10 Cloud service error rate with CAN

Use or not	Filtering of malicious users	SNS	User QoS	Service replication
No	0.29	0.33	0.25	0.20
Yes	**0.11**	**0.13**	**0.10**	**0.19**

either cloud service reliability or the error rate because the replication factor is 1. If the replication factor is increased, service replication will affect the results.

Tables 9 and 10 exhibit average cloud service reliability and the error rate for each scheduling algorithm according to use or non-use. Figures 8 and 9 show the average cloud service reliability and error rate for each case.

5.2.4 Evaluation results with CAN

When considering the four scheduling algorithms for each case, cases 11 and 12 produce the most optimal performance. These cases filter malicious users and use SNS and user QoS. Conversely, the worst performing cases, 1 and 2, neither filter malicious users nor use SNS and user QoS. In other words, filtering malicious users and using SNS and user QoS improve cloud service performance, including cloud service execution time, finish time, reliability, and error rate. Service replication with the high replication factor improves reliability and reduces the error rate.

6 Conclusion

6.1 Summary of the paper

We propose a fault tolerance and QoS scheduling using CAN in MSCC in this paper.

MSCC is a social network-based cloud computing environment supporting user mobility, user QoS, and sharing of cloud services. We use a CAN structure, one of distributed network structures, to manage mobile devices in the computing environment.

Fault tolerance QoS scheduling consists of four subscheduling algorithms: malicious user filtering, cloud service delivery, QoS provisioning, and replication and load-balancing. By using fault tolerance and QoS scheduling, faults arising from mobile device are tolerated, user QoS needs are considered, and members of a social network share cloud services with other members without further authentication.

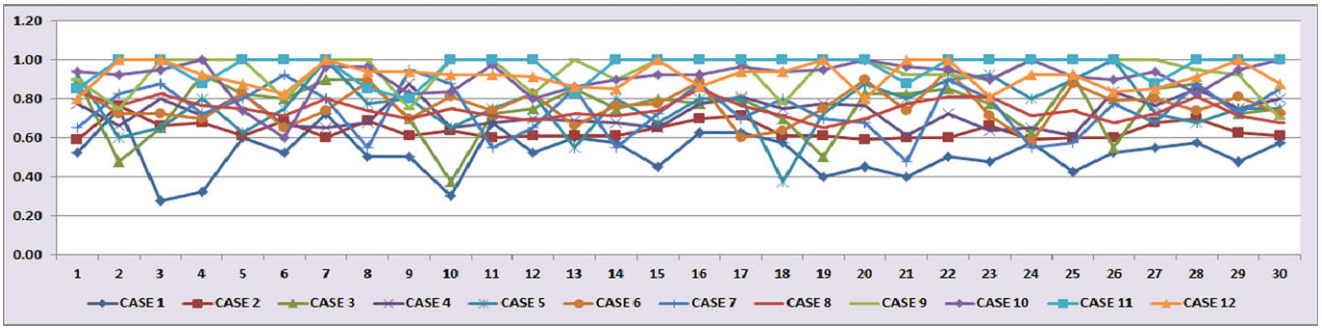

Fig. 8 Cloud service reliability with CAN

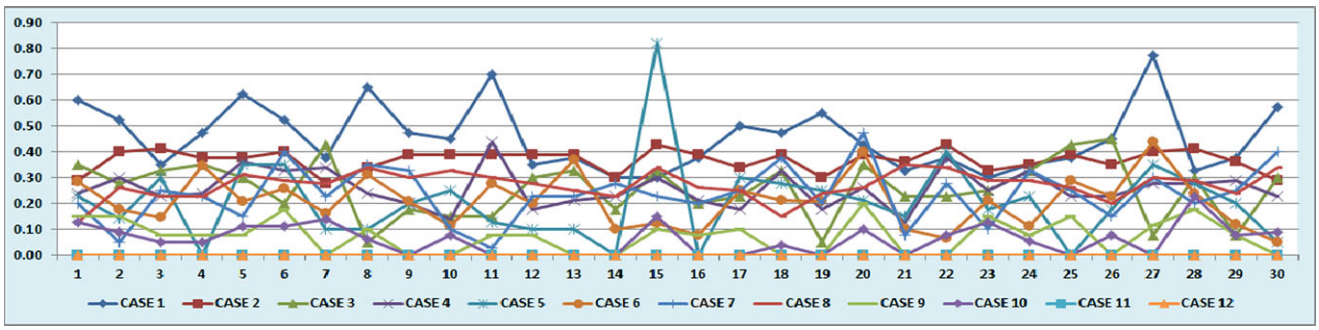

Fig. 9 Cloud service error rate with CAN

As described the simulation results in Sect. 5, using a SNS improves cloud service execution time and service reliability because members of a social network do not act maliciously. Also, filtering malicious users increases cloud service reliability and service replication increases cloud service execution time and reliability.

Therefore, the proposed scheduling algorithms can be applied to improve execution time and reliability in a wide range of computing environments that provide shared services.

6.2 Future work

We plan to study the replication factor for improving cloud service reliability and reducing the error rate when using service replication, as described in Sect. 4.4. In addition, we are planning to conduct a wider variety of experiments to study various QoS metrics and additional factors in MSCC and will apply this fault tolerance QoS scheduling algorithms to real world environments.

References

1. Peter, M., Timothy, G.: The NIST definition of cloud computing. National Institute of Science and Technology, Special Publication 800-145 (2011)
2. http://en.wikipedia.org/wiki/Social_network. Accessed 20 August 2012
3. http://www.ust-global.com/blog/cloud-mobile-social-paradigm.aspx. Accessed 20 August 2012
4. http://www.wired.com/insights/2012/05/social-mobil_e-cloud/. Accessed 20 August 2012
5. Jing, D., Scott, H., Yunghsiang, H., Julia, D.: Fault-tolerant and reliable computation in cloud computing. In: Globecom Workshops, pp. 1601–1605 (2010)
6. Jie, L., Marty, H., You-Wei, C., Youngryel, R.: Fault Tolerance and Scaling in e-Science Cloud Applications: Observations from the Continuing Development of MODISAzure. e-Science 246–253 (2010). doi:10.1109/eScience.2010.47
7. Yilei, Z., Zibin, Z., Michael, L.: BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing. Cloud Computing 444–451 (2011). doi:10.1109/CLOUD.2011.16
8. Yi, H., Bin, G., Fengyu, W.: Cloud model-based security-aware and fault-tolerant job scheduling for computing grid. ChinaGrid, 25–30 (2010)
9. Hyunjoo, K., Sooyong, K., Heon, Y.: Server selection schemes considering node status for a fault-tolerant streaming service on a peer-to-peer network. J. Inf. Process. Syst. **2**(1), 6–12 (2006)
10. Qian, T., Huiyou, C., Yang, Y., Chunqin, G.: A trustworthy management approach for cloud services QoS data. In: ICMLC, pp. 1626–1631 (2010)
11. Habib, S.M., Ries, S., Muhlhauser, M.: Cloud computing landscape and research challenges regarding trust and reputation. In: UIC/ATC, pp. 412–415 (2010)
12. Meng, X., Lizhen, C., Haiyang, W., Yanbing, B.: A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing. In: ISPA, pp. 629–634 (2009)
13. Peng, Z., Zheng, Y.: A QoS-aware system for mobile cloud computing. In: CCIS, pp. 518–522 (2011)

14. Yanchao, Z., Yuguang, F.: A fine-grained reputation system for reliable service selection in peer-to-peer networks. IEEE Trans. Parallel Distrib. Syst. **18**(8), 1134–1145 (2007)
15. Juan, T., Daniel, H., Florin, I., Jes'us, C.: Predictive data grouping and placement for cloud-based elastic server infrastructures. In: CCGrid, pp. 285–294 (2011)
16. Kyle, C., Simon, C., Omer, R., Kris, B.: Social cloud computing: a vision for socially motivated resource sharing. Serv. Comput. **5**(4), 551–563 (2011)
17. Ryan, W., Roger, K., Frank, S., Yan, B., Meeta, S.: Design and implementation of a secure healthcare social cloud system. In: CCGrid, pp. 805–810 (2012)
18. Sylvia, R., Paul, F., Mark, H., Richard, K., Scott, S.: A scalable content-addressable network. In: SIGCOMM, pp. 161–172 (2001)
19. Alexandru, P., David, E., Markus, F., Demetres, K.: Routing in content addressable networks: algorithms and performance. In: IEEE ITC Specialist Seminar (2009)
20. Mohammed, A., Egemen, T., Rui, Z., Lars, K.: Load Balancing for Moving Object Management in a P2P Network. In: DASFAA. LNCS, vol. 4947, pp. 251–266. Springer, Berlin (2008)
21. Sahin, O.D., Gupta, A., Agrawal, D., Abbadi, A.: A peer-to-peer framework for caching range queries. In: ICDE, pp. 165–176 (2004)
22. Amir, B., Anang, H., Muhamad, A., Asad, K.: Under the cloud: a novel content addressable data framework for cloud parallelization to create and virtualize new breeds of cloud applications. In: NCA, pp. 168–173 (2010)
23. Shidong, Z., Bai, W., Gengyu, W., Chao, X.: Web QoS management model based on CAN. In: ISCID, pp. 143–146 (2011)
24. Anuchart, T., Guang, G.: A framework toward a self-organizing and self-healing certificate authority group in a content addressable network. In: WiMob, pp. 614–621 (2010)
25. Henry, R., Daniel, V., Djamshid, T.: Enhancements to CAN for the application as distributed data storage system in grids. In: BroadNets, pp. 432–438 (2005)
26. Roger, Z., Wei-Shinn, K., Haojun, W.: Spatial data query support in peer-to-peer systems. In: COMPSAC, pp. 82–85 (2004)
27. Rodrigo, N.C., Rajiv, R., Anton, B., De Rose, C.A.F., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. SPE J. **41**(1), 23–50. ISSN:0038-0644 (2011)

SookKyong Choi earned her Ph.D. in Computer Science from Korea University in 2013. She is currently a research professor at Korea University researching mobile cloud computing and social cloud computing.

KwangSik Chung received his Ph.D. in Computer Science from Korea University in 2000. He has been a professor in the department of Computer Science at Korea Open National University since 2003. His research areas are grid computing, cloud computing and mobile learning.

Heonchang Yu received his Ph.D. in Computer Science from Korea University in 1994. He has been a professor in the Department of Computer Science Education in Korea University since 1998. His research areas are distributed and cloud computing.