Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Analyzing the service availability of mobile cloud computing systems by fluid-flow approximation*

Hong-wu LV†, Jun-yu LIN, Hui-qiang WANG, Guang-sheng FENG, Mo ZHOU

(*College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China*)

†E-mail: lvhongwu@hrbeu.edu.cn

Received Nov. 28, 2014; Revision accepted June 7, 2015; Crosschecked June 12, 2015

Abstract: Mobile cloud computing (MCC) has become a promising technique to deal with computation- or data-intensive tasks. It overcomes the limited processing power, poor storage capacity, and short battery life of mobile devices. Providing continuous and on-demand services, MCC argues that the service must be available for users at anytime and anywhere. However, at present, the service availability of MCC is usually measured by some certain metrics of a real-world system, and the results do not have broad representation since different systems have different load levels, different deployments, and many other random factors. Meanwhile, for large-scale and complex types of services in MCC systems, simulation-based methods (such as Monte-Carlo simulation) may be costly and the traditional state-based methods always suffer from the problem of state-space explosion. In this paper, to overcome these shortcomings, fluid-flow approximation, a breakthrough to avoid state-space explosion, is adopted to analyze the service availability of MCC. Four critical metrics, including response time of service, minimum sensing time of devices, minimum number of nodes chosen, and action throughput, are defined to estimate the availability by solving a group of ordinary differential equations even before the MCC system is fully deployed. Experimental results show that our method costs less time in analyzing the service availability of MCC than the Markov- or simulation-based methods.

Key words: Service availability, Mobile cloud computing, Fluid-flow approximation, Ordinary differential equations
doi:10.1631/FITEE.1400410 **Document code:** A **CLC number:** TP393

1 Introduction

With excellent performance, low cost, and high degree of usability, cloud computing has become a promising way to manage information for users. It is thought that cloud computing is changing the way that network services are provided. The client typically pays on a per-use basis, rather than maintaining expensive computing hardware. Meanwhile, with the advancement of mobile technology, mobile Internet users have accounted for 70% of the entire Internet users, as an absolute majority. A lot of media applications have migrated to mobile platforms, such as entertainment, health, business, social networking, which bring natural language processing, speech recognition, computer vision, image processing, and other computation- or data-intensive tasks. However, mobile devices have limited computing power, poor storage capacity, and a short battery life, hindering the advancement of this emerging business. Thus, mobile cloud computing (MCC) has been proposed by combining the advantages of mobile computing and cloud computing to overcome these shortcomings (Dinh *et al.*, 2013).

The aim of MCC is to ensure users always obtain continuous, uninterrupted, personalized, and on-demand services; thus, user experience is very important and even determines whether the service can be used continuously. For example, if Bob has stored his data in an MCC system and is often unable to get it, he will change the MCC provider without a doubt. The phenomenon stated above has been approved by a market report from the International Data

* Project supported by the National Natural Science Foundation of China (Nos. 61402127 and 61370212) and the Natural Science Foundation of Heilongjiang Province, China (No. F2015029)
ORCID: Hong-wu LV, http://orcid.org/0000-0002-1917-3978

Corporation (IDC) (Gens *et al.*, 2013). Thus, service availability is usually thought to be very essential for MCC.

However, at present, research on MCC service availability is still in its early stage, and there are some shortcomings:

1. In practice, the availability of MCC is usually guaranteed by service level agreements (SLA), a document defining the relationship between the provider and the recipient. It usually lists many issues, including identifying and defining the customer's needs, providing a framework for understanding complex issues, and reducing areas of conflict (Qi and Gani, 2012). Among these items, the most important is describing the availability of the service to be delivered, which ensures that users obtain good experience. However, the metrics of availability defined by SLA documents are just some fixed boundary conditions, which cannot measure the exact values of the availability of the target system at any given time. At the same time, due to the intrinsic complexity of MCC systems, it is hard to predict the service availability when signing an SLA, especially before the target system is deployed.

2. Currently, some other availability analysis methods for MCC are based on measuring concrete metrics of real-world systems, but the results are not representative since different load levels, different deployment, and other random factors cause a large impact on the measured metrics. So, it is expected to study service availability from a theoretical perspective to reduce random interference.

3. To analyze service availability from a theoretical viewpoint, we usually model MCC by means of state-based methods, for example, the Markov chain or Petri net. However, as the different types of services and the cooperation relationships between components are growing rapidly, traditional modeling methods based on the state space all suffer from the problem of state-space explosion. Many methods have been proposed in recent years to deal with state-space explosion, such as state-space decomposition, state-space aggregation, and fixed-point iteration. Since these methods have to scan the whole state space, the computing time is always an exponential function of the number of all components in the system.

In this paper, instead of measuring the specific metrics of some instances, we model MCC systems using the performance evaluation process algebra (PEPA) (Hillston, 2005) in a formal manner. Based on the model, some critical metrics are defined to estimate and evaluate the service availability of MCC. Furthermore, to overcome the problem of state-space explosion, fluid-flow approximation, a method proposed in recent years (Hillston, 2005), is adopted to convert the PEPA model into a group of ordinary differential equations (ODEs). Unlike traditional state-based analytical techniques, the fluid-flow approximation method involves approximating the underlying discrete state space with continuous real-valued variables and describing the time-evolution of those variables with ODEs (Hayden and Bradley, 2008). In other words, it needs only to know the population of each species of components, rather than the states of all the components. So, the number of states that it supports is over 10100. In addition, the proposed model can be used even before the target system is deployed.

2 Modeling mobile cloud computing based on PEPA

2.1 PEPA and fluid-flow approximation

PEPA, a classical process algebra, is known for its easy use (Hillston, 2005). Besides model checking, it can be used for quantitative analysis. Traditionally, PEPA is mapped to an underlying Markov chain for the analysis of performance, but analyzers usually encounter state-space explosion problems, as the number of states is too large. In this study, a new definition of PEPA is introduced using fluid-flow approximation proposed by Hillston (2005).

The state of the PEPA model at time t can be represented by $P(t)$, which has the following characteristics:

$$P(t) ::= P(t) \underset{L}{\triangleright\!\triangleleft} P(t) \mid Y(t), \tag{1}$$

$$P(t) ::= (\alpha, r).P \mid P + P \mid P / L \mid A, \tag{2}$$

$$Y(t) ::= P(t) \parallel P(t) \parallel \ldots P(t), \tag{3}$$

where $P(t) \underset{L}{\triangleright\!\triangleleft} P(t)$ and $Y(t)$ represent interaction and parallel actions, respectively. The choice +, cooperation $\underset{L}{\triangleright\!\triangleleft}$, hiding /, and constant definition A are

the basic syntaxes. In the prefix (α, r), r means the apparent rate of action α. For details of the operation, one can refer to Hillston (2005) and Bradley *et al.* (2008).

Based on the new definition of PEPA, fluid-flow approximation can be used to convert PEPA into ODEs. According to Castiglione *et al.* (2014), the nature of fluid-flow approximation is that the large number of discrete states is considered to be of continuous change. Then some ODEs can be built to describe the trend of state changes.

Given n classes of components, the ith class of components is denoted as $\boldsymbol{C}_i$. Each class of components has a series of derivations, and the jth derivation of $\boldsymbol{C}_i$ is called C_{ij}. Let $N(C_{ij},t)$ denote the number of components at time t. Exit(C_{ij}) and Enter(C_{ij}) represent the sets of exit and entry activities of a local derivation, respectively. In a short time Δt, the changes of an arbitrary derivation C_{ij} can be described as follows:

$$\begin{aligned} &N(C_{ij},t+\Delta t)-N(C_{ij},t)\\ &=\underbrace{-\sum_{\alpha\in \mathrm{Exit}(C_{ij})}\rho_\alpha(C_{ij},P(t))\Delta t}_{\text{Exit activities}}+\underbrace{\sum_{\beta\in \mathrm{Enter}(C_{ij})}\rho_\beta(C_{ik},P(t))\Delta t}_{\text{Enter activities}}, \end{aligned} \tag{4}$$

where the component rate $\rho_\alpha(C_{ij}, P(t))$ captures the local effect of $P(t)$ on component C_{ij}, and $k\neq j$. Then a real-valued variable $v_{ij}(t)$ is used to approximate the discrete variable $N(C_{ij}, t)$, denoted as $v_{ij}(t)=E[N(C_{ij}, t)]$. Furthermore, according to Hayden *et al.* (2012), the error between $v_{ij}(t)$ and $N(C_{ij}, t)$ will tend to 0 while $\sum_{i,j}N(C_{ij},t)\to\infty$.

Let Δt be close to 0. We obtain

$$\begin{aligned} \frac{\mathrm{d}v_{ij}(t)}{\mathrm{d}t}=&-\sum_{\alpha\in \mathrm{Exit}(C_{ij})}\rho_\alpha(C_{ij},P(t))+\sum_{\beta\in \mathrm{Enter}(C_{ij})}\rho_\beta(C_{ik},P(t)),\\ &j=1,2,\ldots,|C_i|,\ i=1,2,\ldots,n. \end{aligned} \tag{5}$$

For details, one can refer to Tribastone *et al.* (2012a). Therefore, the complexity of solving the model is related only to the number of the types of components, rather than the population of all types of components.

2.2 MCC model based on PEPA

Currently, there are many types of MCC applications in our daily lives, and the most famous examples include 'searching the lost child' (Satyanarayanan, 2011), 'the translation in museum' (Huerta-Canepa and Lee, 2010), 'disaster relief' (Fernando *et al.*, 2013), and 'the traffic congestion map' (Rishabh *et al.*, 2013). These scenes basically have the same mode, and in this study, the traffic congestion map is chosen as an example to analyze the service availability of MCC.

Example 1 (Traffic congestion map) While a driver is on his/her way to the airport, he/she sends a request to a cloud system by phone to get a traffic congestion map and avoid encountering a traffic jam. Then the cloud gathers the urban traffic status in real time through a lot of taxis distributed in the city, and forms a near real-time traffic congestion map after processing the data collected. Then the map is returned to the driver for choosing the best road.

To establish a model of universal significance, it is necessary to ignore the details not directly associated with the service availability of MCC. Taking the traffic congestion map as an example, it is easy to find that an MCC system has the following typical characteristics:

1. An MCC system can be divided into two parts, the cloud and a set of mobile devices. The former is either a traditional cloud computing system or a set of computing nodes closely related.

2. While the cloud receives a service request from a user, every mobile device is awakened to collect information.

3. The data collected is submitted to the cloud through a wireless link and computed by means of classic cloud computing.

4. The final results will be returned to the users.

As stated above, computation- or data-intensive tasks in an MCC system are submitted to the cloud instead of being handled in local devices. This is usually called 'offloading computing' (Ou *et al.*, 2007). Different from traditional interactive computing, after submitting a request, a user will not interact with the cloud again until a final result is returned. So, we primarily describe the MCC system from the cloud and the mobile device, ignoring user behaviors.

2.2.1 Modeling the cloud

Without loss of generality, we assume that the cloud part is based on a map-reduce structure whose main steps include requesting & waiting, splitting, mapping, shuffling, and reducing (Fig. 1).

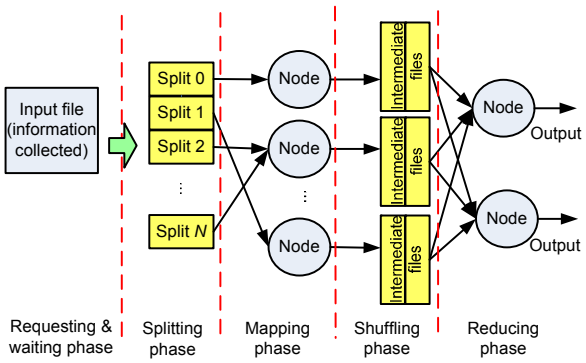

Fig. 1 Process of cloud computing

Assuming the sojourn time of each action follows an exponential distribution in the process of cloud computing (Chilwan *et al.*, 2011; Longo *et al.*, 2011), PEPA can be used to model this process (Fig. 2). Certainly, every step may have a more complex process and contains many details, but PEPA supports a hierarchical structure and can be easily described by combining some simple sentences (Fourneau *et al.*, 2002). Therefore, we ignore the details, making the model as simple as possible.

Master=(broadcast_requesting, r_1).Waiting;
Waiting=(return, r_2).Splitting;
Splitting=(split, r_3).Mapping;
Mapping=(map, r_4).Shuffling;
Shuffling=(shuffle, r_5).Reducing;
Reducing=(reduce, r_6).Master;

Fig. 2 The PEPA model of the cloud

Assume that the cloud has the ability to process M tasks in parallel, which means the tasks can be divided into M pieces at most. That is, the number of Mappings is M at most. Thereafter, the component Master as a derivation of Mapping also has a maximum amount M. Then the cloud part is described as

$$\text{Master}[M].$$

2.2.2 Modeling the devices

For the mobile devices, a certain port is always monitored to wait for the command of collecting information. As soon as it receives a request from the cloud, the device turns to the state of Awake, and begins to gather data with some sensors. Afterwards, the mobile device changes to the Sensing state until the information collected is submitted. While finishing a circle as stated above, the mobile device goes back to the state of Device and gets ready for the next service request. The devices are modeled as shown in Fig. 3.

Device=(broadcast_requesting, w_1).Awake;
Awake=(sensoring, w_2).Sensing;
Sensing=(return, w_3).Device;

Fig. 3 The PEPA model of the devices

Given that there are N mobile devices in the target system, the Device part can be described as

$$\text{Device}[N].$$

Therefore, a simple MCC model can be described by the interaction between the cloud part and the device part. If the parallel computing capability of the cloud is M and the number of mobile devices is N, then the MCC model is built as follows:

$$\text{Master}[M] \underset{L}{\triangleright\!\triangleleft} \text{Device}[N], \tag{6}$$

where L is a set of cooperation actions, L={broadcast_requesting, return}. Furthermore, considering a more complex scene where there are many types of devices, it can be rewritten as

$$\text{Master}[M] \underset{L}{\triangleright\!\triangleleft} (\text{Device}_1[N_1] \| \text{Device}_2[N_2] \| \ldots \| \text{Device}_k[N_k]),$$

where $\sum_{i=1}^{k} N_i = N$, and N_i represents the number of the ith kind of device.

3 Metrics of availability

Just like Software-as-a-Service (SaaS), the service availability of MCC is characterized by some key metrics. In this section, we will discuss four of the most important metrics of service availability, including response time of service, minimum sensing time of devices, minimum number of nodes chosen, and action throughput.

3.1 Response time of service

Response time of service (RTS), one of the core

metrics of service availability, directly determines the user experience of MCC. For the characteristics inherited from mobile computing, mobile devices may move into an area with no signal, or suffer from masking or channel interference. These problems greatly increase the RTS, and even lead to a terminal timeout.

Traditionally, for MCC, the average RTS usually relates to the average processing time of all types of services running on the same platform. However, it is not very accurate in practice for a certain service as the execution time of each service varies greatly. In this study, RTS for a single type of service, or SRTS, is investigated.

As stated in Section 2.2, the service of MCC is implemented primarily in the cloud part by means of offloading computing. For a certain type of service, SRTS refers to the shortest possible time from receiving the service request to returning the final results for the first time.

Combined with the model given in Section 2.2, we describe this metric in a formal manner. SRTS can be treated as the passage time to complete the first cycle from the state of Master, that is, Master→Reduce→Master. Then we can modify the latter Master component and introduce a new component, Master′. Correspondingly, the cloud part of MCC can be rewritten as shown in Fig. 4. Based on the model, we can define the SRTS using the passage time which is an infimum, the shortest time used from the source state to reach a set of target states (Hayden *et al.*, 2012).

Master=(broadcast_requesting, r_1).Waiting;
Waiting=(return, r_2).Splitting;
Splitting=(split, r_3).Mapping;
Mapping=(map, r_4).Shuffling;
Shuffling=(shuffle, r_5).Reducing;
Reducing=(reduce, r_6).Master′;
Master′=(broadcast_requesting, r_1).Waiting′;
Waiting′=(return, r_2).Splitting′;
Splitting′=(split, r_3).Mapping′;
Mapping′=(map, r_4).Shuffling′;
Shuffling′=(shuffle, r_5).Reducing′;
Reducing′=(reduce, r_6).Master′;

Fig. 4 The revised model of the cloud part

Actually, a component stays in a certain state at time t with a probability. Similarly, a component arrives at the target collection with a probability $\theta(t)$ at time t. As a matter of fact, what we are really concerned with is which t makes $\theta(t)$ greater than θ_0. The parameter θ_0 is a threshold indicating that a component has reached the target collection. It has a close relationship with the specific type of service. Given the target set C'={Master′, Waiting′, Splitting′, Mapping′, Shuffling′, Reducing′}, the number of all components in C' is $N(C', t)$. While there are M Masters in the system, to complete the task, the value of $N(C', t)$ must be more than $M\times\theta_0$. Thus, we can define the SRTS as follows:

Definition 1 For any $t>0$, $N(C', t)$ represents the number of components in C' at time t. Then,

$$\text{SRTS}=\inf\{t, N(C', t)\geq M\times\theta_0\}, \tag{7}$$

where $N(C', t)$ can be expressed as

$$N(C', t)=\sum_{\text{derivation}\in C'} E[N(\text{derivation}, t)], \tag{8}$$

with $E[N(\text{derivation}, t)]$ representing the fluid-flow approximation of the number of component derivations at time t. All of the items in the right-hand side of the above equations can be obtained by solving the model in this subsection. A detailed description will be given in Section 4.1.

3.2 Minimum sensing time of devices

The data collected by mobile devices is the basis of MCC. However, due to the unstable wireless link between mobile devices and the cloud system, terminals may stay in an area with no signal or suffer from channel interference. This phenomenon often leads to a device timeout; that is, the cloud part always waits for a response from the device after sending some commands. To avoid meaningless waiting, the analyzer needs to estimate the minimum sensing time used by devices.

First we discuss the definition of the minimum sensing time. Take the traffic congestion map as an example. The process of sensing begins with receiving a command from the cloud, and ends with returning the data collected. So, the minimum sensing time (MST) of the device is the least time used by devices from receiving a command to returning a final result.

Consider there is only one type of device in the system. Therefore, MST is the time used to complete the process of Device→Sensing→Device. Like in

Fig. 4, to distinguish the components, the latter Device is denoted as Device′. Then the revised model of the device part is as shown in Fig. 5.

Device=(broadcast_requesting, w_1).Awake;
Awake=(sensoring, w_2).Sensing;
Sensing=(return, w_3).Device
Device′=(broadcast_requesting, w_1).Awake′;
Awake′=(sensoring, w_2).Sensing′;
Sensing′=(return, w_3).Device′;

Fig. 5 The revised model of the device part

For simplicity, we assume that all selected devices are identical, and that the delay time of the actions is independent and identically distributed. When all mobile devices are equipped with the same MCC client, this assumption is reasonable. Then we can give a definition of MST.

Definition 2 For any device i, $1\leq i\leq N$, where N is the overall number of devices, record the current state of device i as S_i. Then MST can be seen as the shortest time from the source state of Device to the target set of S'={Device′, Awake′, Sensoring′}:

$$\text{MST}=\inf\{t, S_i\in S'\}. \tag{9}$$

According to the above definition, MST is hard to compute in practice. We will look for a way to quantify it.

According to the research on individual passage time (Hayden *et al.*, 2012), up to time t, finishing a circle for a device means the device belongs to S', so the probability of finishing a circle is

$$\begin{aligned}P\{S_i(t)\in S'\}=P\{&S_i(t)\neq\text{Device}, S_i(t)\neq\text{Awake},\\ &S_i(t)\neq\text{Sensing}\}.\end{aligned} \tag{10}$$

According to the above statement which means all selected devices are identical, we have

$$\begin{aligned}E[N(\text{Device},t)]&=\sum_{i=1}^{N}E\left[1_{\{S_i=\text{Device}\}}\right]\\ &=N\cdot P\{S_i=\text{Device}\}.\end{aligned} \tag{11}$$

Then we can obtain a theorem about the metric MST:

Theorem 1 For any time $t>0$, the probability of MST follows

$$\begin{aligned}P\{\text{MST}\leq t\}=1-\frac{1}{N}\big(&E[N(\text{Device},t)]\\ &+E[N(\text{Awake},t)]+E[N(\text{Sensing},t)]\big).\end{aligned} \tag{12}$$

Proof According to Eq. (10), if $t>$MST, all the components have finished a circle. Thus,

$$\begin{aligned}P\{\text{MST}\leq t\}=&P\{S_i(t)\neq\text{Device}, S_i(t)\neq\text{Awake},\\ &S_i(t)\neq\text{Sensoring}\}\\ =&1-P\{S_i(t)=\text{Device}\}-P\{S_i(t)=\text{Awake}\}\\ &-P\{S_i(t)=\text{Sensing}\}.\end{aligned}$$

Owing to Eq. (11), we have

$$\begin{aligned}P\{\text{MST}\leq t\}=1-\frac{1}{N}\big(&E[N(\text{Device},t)]\\ &+E[N(\text{Awake},t)]+E[N(\text{Sensing},t)]\big).\end{aligned}$$

3.3 Minimum number of nodes chosen

Due to the statements in Section 3.2, the infrastructure of the MCC system is usually unreliable. To deal with unexpected faults and random high loads, each service is always allocated with redundancy processing capabilities. Therefore, more nodes than the minimum number of devices needed are deployed to provide normal services. In reality, requirements will be sent by the cloud to a lot of local devices in this area. However, the fact that fewer devices are needed means less energy consumption and fewer links unexpectedly interrupted. Thus, it is important to estimate the minimum number of nodes needed and determine the range of devices chosen.

Assuming that the expected minimum number of nodes is N_0, and that the proportion of redundant sources is γ, then the actual allocation of devices is $(1+\gamma)\times N_0$. The degree of redundancy is determined by a variety of environmental factors. According to Section 3.2, we can analyze this metric through the probability of a terminal timeout.

With respect to Theorem 1, it is easy to obtain the probability of terminal timeout by removing the proportion of no timeout.

Lemma 1 Let the expected minimum number of nodes be N_0, $t>0$. If a node is a timeout at a certain time t, the probability of the terminal timeout is

$$\begin{aligned}\lambda(t)=\frac{1}{N_0}\big(&E[N(\text{Device},t)]+E[N(\text{Awake},t)]\\ &+E[N(\text{Sensing},t)]\big).\end{aligned} \tag{13}$$

Based on Lemma 1, we can give a definition of the minimum number of nodes needed, given that the proportion of a device timeout is $\lambda(t)$, the minimum number of nodes needed is N_0, and the minimum number of nodes chosen is N_1. They satisfy the following relationship:

$$N_0=N_1\times(1-\lambda(t)). \tag{14}$$

Combining Eqs. (13) and (14), we can obtain the following theorem:

Theorem 2 Given $t>0$, the minimum number of nodes needed N_0, the number of devices chosen N_1 satisfies

$$N_1=N_0^2\times\big(N_0-(E[N(\text{Device},t)]+E[N(\text{Awake},t)]+E[N(\text{Sensing},t)])\big)^{-1}. \tag{15}$$

Certainly, N_1 is less than the maximum number of devices that can be reached within the area; that is, $N_1\leq(1+\gamma)\times N_0$. This metric is closely related to energy consumption and network interruption. It also plays an essential role in MCC. Furthermore, the analysis results can be treated as evidence for increase in the number of redundant devices.

3.4 Action throughput

Throughput is a classical metric to assess the transaction processing capability of a component. A better processing capability also means good service availability under the same conditions; that is, the user has a greater probability of gaining what he/she wants. Action throughput is a new performance metric developed based on throughput, mainly assessing the frequency of a type of action implemented (Satyanarayanan, 2011). In an MCC system, action throughput is also a very important metric. Taking the case of the traffic congestion map as an example, the throughput of broadcast_requesting denotes the ability of MCC to send service requests per unit of time. The greater the value is, the more transactions are handled. Similarly, the throughput of return represents the frequency of returning service results per unit of time. While the throughput increases, the service processing abilities become better. In this study, we focus on the throughput of these two actions.

First, considering one of the simplest cases, that is, action α is not a cooperation action between components in the PEPA model.

Lemma 2 Let k indicate the kth component, Act(k) the action set triggered by k, r the apparent rate of action α, and π_k the steady-state probability of component k. We have

$$\text{Throughput}(\alpha)=\sum_k\left(\pi_k\sum_{(\alpha,r)\in\text{Act}(k)}r\right). \tag{16}$$

Second, for a cooperation action, there is more than one partner deciding the action throughput. For example, the action throughput of broadcast_requesting is determined by both Device and Master. According to Tribastone *et al.* (2012b), we can redefine the action throughput in a new format:

Lemma 3 Let $L=\{C_1, C_2, \ldots, C_q\}$, $q\in\mathbb{Z}^*$, $q\geq2$, and all of the elements of L have a cooperation action α. The action throughput of α in a local place L can be described as

$$\text{Throughput}_L(\alpha)=\min_{k\in L}\left(\pi_k\sum_{(\alpha,r)\in\text{Act}(k)}r\right). \tag{17}$$

Owing to the two statements above, we can give a more general result of action throughput:

Theorem 3 In a PEPA model, all components constitute a set $C=L\cup\overline{L}$, and all of the elements of L have a cooperation action α. The action throughput of α can be described as

$$\text{Throughput}(\alpha)=\sum_{k\in\overline{L}}\left(\pi_k\sum_{(\alpha,r)\in\text{Act}(k)}r\right)+\min_{l\in L}\left(\pi_l\sum_{(\alpha,r)\in\text{Act}(l)}r\right). \tag{18}$$

While there are a large number of components, solving the solution of π_k will suffer from state-space explosion, so the fluid-flow approximation method is used for analysis of action throughput. Correspondingly, Eq. (18) becomes

$$\text{Throughput}(\alpha)=\sum_{k\in\overline{L}}\left(E[N(C_k)]\sum_{(\alpha,r)\in\text{Act}(k)}r\right)+\min_{l\in L}\left(E[N(C_l)]\sum_{(\alpha,r)\in\text{Act}(l)}r\right), \tag{19}$$

where $E[N(C_k)]$ represents the mathematical expectation of the number of components, k, after reaching a steady state.

4 Analyzing the metrics of service availability by fluid-flow approximation

According to the metrics discussed above, we analyze the service availability with a simple case of the traffic congestion map. The parameters are chosen as shown in Table 1.

Table 1 Default parameters used in the model

Parameter	Value	Parameter	Value
r_1	0.2	r_6	0.1
r_2	0.5	w_1	0.2
r_3	1	w_2	2
r_4	1	w_3	1
r_5	2		

It is necessary to point out that the values of the parameters are computed using the classical method proposed by Huang *et al.* (1995). Given that the average delay of action α is T, the value of α is $1/E(T)$, where $E(T)$ is the mathematical expectation of T. For instance, the action of broadcast_requesting has a long latency owing to terminal timeout or the mechanism of retransmission. Then its mathematical expectation of the average delay is 5 time units, and r_2 is 1/5. Other parameters have similar semantics.

Next, we will analyze the metrics of service availability and the impacts of the parameters. In this study, the fluid-flow approximation method is adopted, as there are a lot of mobile devices in MCC systems and accordingly the state space of the PEPA model is very large.

4.1 Analysis of SRTS

Before analyzing SRTS, it is necessary to solve the PEPA model in Section 3.1. We convert the model into ODEs by fluid-flow approximation. Due to Hillston (2005), the activity diagram of this model is as shown in Fig. 6.

For easy presentation, we use the following mapping:

$N(\text{Master}, t)\to v_{11}$, $N(\text{Waiting}, t)\to v_{12}$,
$N(\text{Splitting}, t)\to v_{13}$, $N(\text{Mapping}, t)\to v_{14}$,
$N(\text{Shuffling}, t)\to v_{15}$, $N(\text{Reduce}, t)\to v_{16}$,
$N(\text{Master}', t)\to v_{111}$, $N(\text{Waiting}', t)\to v_{121}$,
$N(\text{Splitting}', t)\to v_{131}$, $N(\text{Mapping}', t)\to v_{141}$,
$N(\text{Shuffling}', t)\to v_{151}$, $N(\text{Reduce}', t)\to v_{161}$,
$N(\text{Device}, t)\to v_{21}$, $N(\text{Awake}, t)\to v_{22}$,
$N(\text{Sensing}, t)\to v_{23}$.

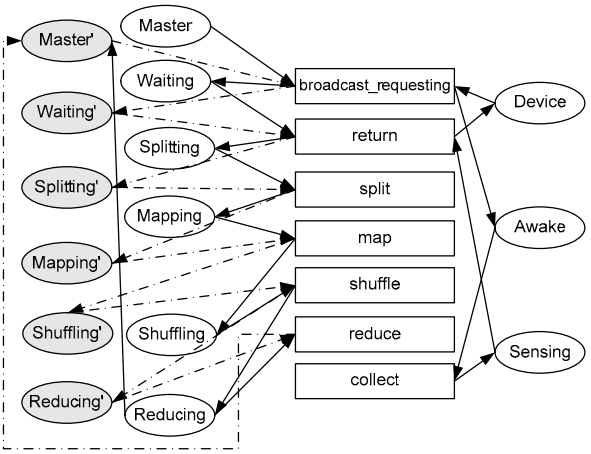

Fig. 6 Activity diagram for the revised model of the cloud part

Using Eq. (5), the ODEs are obtained as given in Eq. (20).

Assuming the initial population of components is (99.5, 0.1, 0.1, 0.1, 0.1, 0.1, 0, 0, 0, 0, 0, 0, 0, 100, 0, 0, 0), the state space is $12^{100}\times3^{100}>10^{100}$ while we use a traditional Markov chain method before simplification. Clearly, it will bring a state-space explosion. In contrast, it is easy to obtain the solution of the case using Eq. (20). As shown in Fig. 7, we can find the population of each of the components in the set C' after the first ground of service.

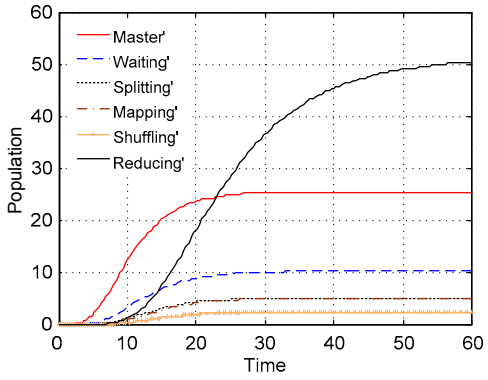

Fig. 7 The number of components in set C'

$$
\begin{cases}
v'_{11} = -\dfrac{v_{11}}{v_{11}+v_{111}}\min(r_1(v_{11}+v_{111}), w_1 v_{21}),\\
v'_{12} = -\dfrac{v_{12}}{v_{12}+v_{121}}\min(r_2(v_{12}+v_{121}), w_3 v_{23})\\
\qquad +\dfrac{v_{11}}{v_{11}+v_{111}}\min(r_1(v_{11}+v_{111}), w_1 v_{21}),\\
v'_{13} = -r_3 v_{13} + \dfrac{v_{12}}{v_{12}+v_{121}}\min(r_2(v_{12}+v_{121}), w_3 v_{23}),\\
v'_{14} = -r_4 v_{14} + r_3 v_{13},\\
v'_{15} = -r_5 v_{15} + r_4 v_{14},\\
v'_{16} = -r_6 v_{16} + r_5 v_{15},\\
v'_{111} = -\dfrac{v_{111}}{v_{11}+v_{111}}\min(r_1(v_{11}+v_{111}), w_1 v_{21}) + r_6 v_{16},\\
v'_{121} = -\dfrac{v_{121}}{v_{12}+v_{121}}\min(r_2(v_{12}+v_{121}), w_3 v_{23})\\
\qquad +\dfrac{v_{111}}{v_{11}+v_{111}}\min(r_1(v_{11}+v_{111}), w_1 v_{21}),\\
v'_{131} = -r_3 v_{131} + \dfrac{v_{121}}{v_{12}+v_{121}}\min(r_2(v_{12}+v_{121}), w_3 v_{23}),\\
v'_{141} = -r_4 v_{141} + r_3 v_{131},\\
v'_{151} = -r_5 v_{151} + r_4 v_{141},\\
v'_{161} = -r_6 v_{161} + r_5 v_{151},\\
v'_{21} = -\min(r_1(v_{11}+v_{111}), w_1 v_{21})\\
\qquad +\min(r_2(v_{12}+v_{121}), w_3 v_{23}),\\
v'_{22} = -w_2 v_{22} + \min(r_1(v_{11}+v_{111}), w_1 v_{21}),\\
v'_{23} = -\min(r_2(v_{12}+v_{121}), w_3 v_{23}) + w_2 v_{22}.
\end{cases}
\tag{20}
$$

Based on the solution of the ODEs above, the plot of SRTS is obtained (Fig. 8), where the *X*-axis represents SRTS and the *Y*-axis denotes the proportion of computing nodes that have finished their jobs. As SRTS increases, more and more computing nodes complete their assigned tasks. For example, if θ_0=0.9519, then SRTS=41.97. Therefore, given θ_0 it is easy to obtain any SRTS by the solution of the ODEs.

In this case, to avoid a 0-division calculation, we assume that initially there are a few components of Waiting, Splitting, Mapping, Shuffling, and Reducing, but in reality the number of these components should be close to 0. In the next case, the impacts of the initial conditions on SRTS will be analyzed.

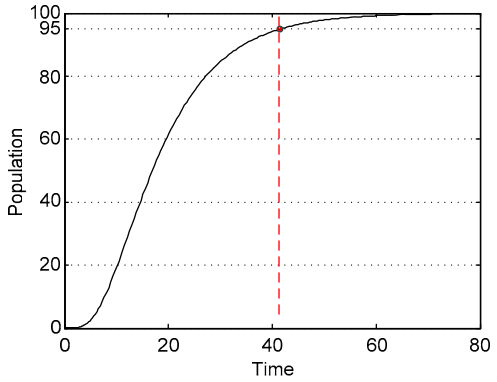

Fig. 8 The number of computing nodes that have arrived at the target collection (SRTS=41.97 when θ_0= 0.9519)

For comparison, the total number of components remains unchanged, and the initial number of each of the components is separately set to (99.5, 0.1, 0.1, 0.1, 0.1, 0.1, 0, 0, 0, 0, 0, 0, 100, 0, 0), (99.95, 0.01, 0.01, 0.01, 0.01, 0.01, 0, 0, 0, 0, 0, 0, 100, 0, 0), (95, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 100, 0, 0). Fig. 9 gives the corresponding curves under these three different initial conditions. The results of SRTS are not very different. While the time is no more than 40, the curve in the 3rd initial condition is slightly higher than the others. The curves with the other two initial conditions almost overlap with each other everywhere. These results show that the initial condition has little impact on SRTS.

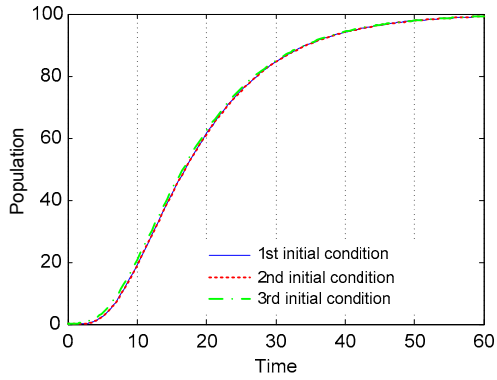

Fig. 9 Impact of the initial condition on the response time of service

4.2 Analysis of MST

Just like the analysis of SRTS in Section 4.1, before computing MST, the PEPA model in Section 3.2 needs to be converted into ODEs. First, the

relationships of the components of this model are given in Fig. 10.

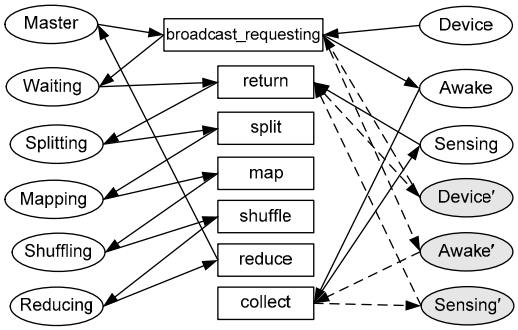

Fig. 10 The activity diagram for the revised model of the device part

The corresponding mapping is then shown as

N(Master)→v_{11}, N(Waiting)→v_{12},
N(Splitting)→v_{13}, N(Mapping)→v_{14},
N(Shuffling)→v_{15}, N(Reduce)→v_{16},
N(Device)→v_{21}, N(Awake)→v_{22},
N(Sensing)→v_{23}, N(Device′)→v_{211},
N(Awake′)→v_{221}, N(Sensing′)→v_{231}.

According to the definition of fluid-flow approximation given by Eq. (5), the corresponding ODEs are obtained as given in Eq. (21).

Assuming the initial number of components is (100, 0, 0, 0, 0, 0, 99.8, 0.1, 0.1, 0, 0, 0), we can obtain the solution to this case using Eq. (21). Fig. 11 shows the number of components for S' after the first ground of service. It is observed that these curves will stabilize and converge to a fixed value after the 150th time unit.

Fig. 12 gives the cumulative probability density function (CDF) of MST as determined by Eq. (12). For instance, setting the threshold of MST to be 100, then the CDF of MST is 0.9399. As time goes on, the CDF of MST gradually increases. It means that more devices finish a circle while the threshold of MST increases. In other words, the number of components in S' grows with increase of MST. Furthermore, as the threshold increases, most of the devices have finished sensing, the growth of the number of components in S' slows down, and finally all of the components have entered S'.

$$
\begin{cases}
v'_{11} = -\min(r_1 v_{11}, w_1(v_{21}+v_{211})) + r_6 v_{16}, \\
v'_{12} = -\min(r_2 v_{12}, w_3(v_{23}+v_{231})) \\
\qquad + \min(r_1 v_{11}, w_1(v_{21}+v_{211})), \\
v'_{13} = -r_3 v_{13} + \min(r_2 v_{12}, w_3(v_{23}+v_{231})), \\
v'_{14} = -r_4 v_{14} + r_3 v_{13}, \\
v'_{15} = -r_5 v_{15} + r_4 v_{14}, \\
v'_{16} = -r_6 v_{16} + r_5 v_{15}, \\
v'_{21} = -\dfrac{v_{21}}{v_{21}+v_{211}} \min(r_1 v_{11}, w_1(v_{21}+v_{211})), \\
v'_{22} = -w_2 v_{22} + \dfrac{v_{21}}{v_{21}+v_{211}} \min(r_1 v_{11}, w_1(v_{21}+v_{211})), \\
v'_{23} = -\dfrac{v_{23}}{v_{23}+v_{231}} \min(r_2 v_{12}, w_3(v_{23}+v_{231})) + w_2 v_{22}, \\
v'_{211} = -\dfrac{v_{211}}{v_{21}+v_{211}} \min(r_1 v_{11}, w_1(v_{21}+v_{211})) \\
\qquad + \min(r_2 v_{12}, w_3(v_{23}+v_{231})), \\
v'_{221} = -w_2 v_{221} + \dfrac{v_{211}}{v_{21}+v_{211}} \min(r_1 v_{11}, w_1(v_{21}+v_{211})), \\
v'_{231} = -\dfrac{v_{231}}{v_{23}+v_{231}} \min(r_2 v_{12}, w_3(v_{23}+v_{231})) + w_2 v_{221}.
\end{cases}
\tag{21}
$$

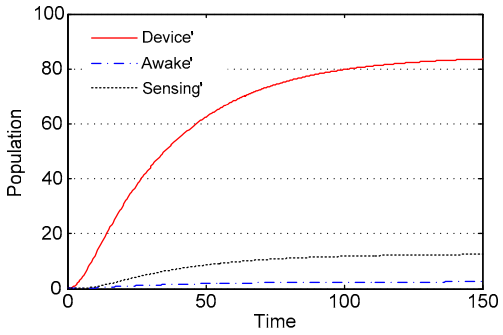

Fig. 11 The number of components in set S'

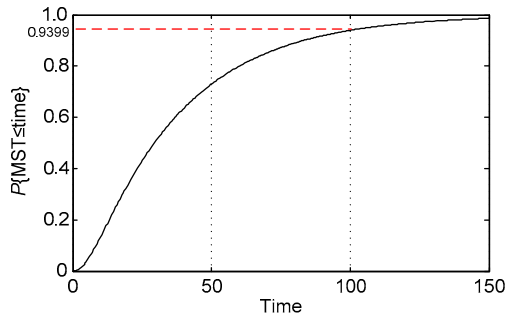

Fig. 12 The cumulative probability density function of the minimum sensing time

4.3 Estimating the minimum number of nodes chosen

According to Eq. (14), it is necessary to compute the probability of terminal timeout before analyzing the minimum number of nodes chosen. Given that the initial number of each of the components is set to (100, 0, 0, 0, 0, 0, 99.8, 0.1, 0.1, 0, 0, 0) as in Section 4.2, the probability of terminal timeout can be determined (Fig. 13).

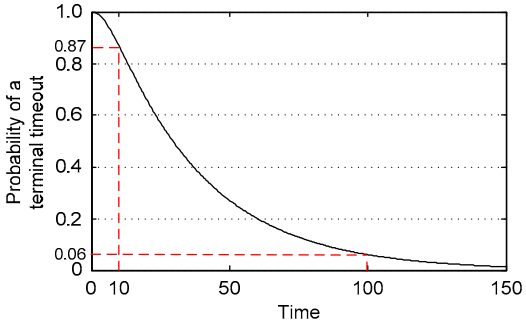

Fig. 13 The probability of a terminal timeout

As shown in Fig. 13, the probability of a terminal timeout becomes lower and lower while the threshold of the timeout grows. Supposing that the threshold is set to be 10 time units, the probability of terminal timeout is 0.87 according to Eq. (13). Furthermore, as the threshold is set to be 100 units, the probability is near 0.06, which is low enough to ensure sufficient devices finishing the first round of information collection. Similar to SRTS, the probability of a terminal timeout is easily obtained after setting a certain threshold.

According to Eq. (15), we can obtain the minimum number of nodes chosen as shown in Fig. 14. As the timeout threshold of the terminal increases, much fewer nodes need to be deployed. With the increase of the threshold, the minimum number of nodes needed converges to 100. It is certain that the number of nodes needed is equal to N_0 when there is no terminal timeout. To sum up, as soon as the threshold of timeout is given, it is easy to obtain the minimum number of nodes chosen in the MCC system.

4.4 Computing the action throughputs

The action throughputs can be computed according to Eq. (19). For example, when the values of the parameters are assigned as in Table 1, the action throughputs of broadcast_requesting and return are computed as follows:

Throughput(broadcast_requesting)
$=\min(E[N(\text{Master}, t)]\times r_1, E[N(\text{Device}, t)]\times w_1)$
$=4.26$,
Throughput(return)
$=\min(E[N(\text{Waiting}, t)]\times r_2, E[N(\text{Sensing}, t)]\times w_3)$
$=2.13$.

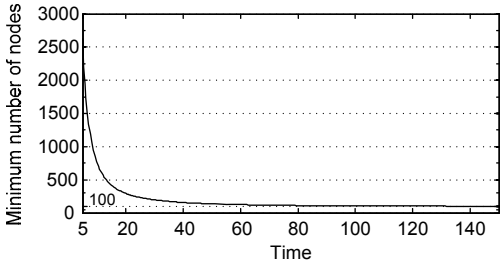

Fig. 14 The minimum number of devices chosen

In the above case, both broadcast_requesting and return are cooperation actions, whose implementation parameters are determined by both the cloud and the devices. So, we should change the two sides when increasing the action throughputs.

To check the impacts of broadcast_requesting from parameters on the action throughputs, the critical parameters r_1 and w_1 must be studied. First, we change r_1 from 0.2 to 10. Table 2 shows the changes of the action throughput of broadcast_requesting.

Table 2 shows that the action throughput of broadcast_requesting increases very slowly as r_1 grows. Moreover, according to the meaning of $1/r_1$, the mathematical expectation of the time interval between two sending requests, the value of r_1 must fall in a certain range; that is, the action throughput will not become too large.

Similarly, the other parameter w_1 changes from 0.03 to 0.1 when r_1=0.2 (Table 3). The action throughput of broadcast_requesting is reduced when w_1 decreases. This is because the time interval between two sending requests becomes too long for the devices. Thus, the cloud does not have enough data to deal with, while the value of w_1 is too small, and the action throughput of broadcast_requesting decreases. Meanwhile, the service availability of MCC will be degraded to a low level.

Table 2 The action throughput of broadcast_requesting while r_1 changes

r_1	Population		Action throughput
	Master	Device	
0.2	21.3	74.3	4.26
0.3	15.3	72.4	4.59
0.4	11.9	71.3	4.77
0.6	8.3	70.1	4.97
0.8	6.3	69.5	5.07
1.0	5.1	69.1	5.12
2.0	2.6	68.6	5.27
3.0	1.8	68.0	5.32
5.0	1.1	67.6	5.36
10.0	0.5	67.6	5.39

Table 3 The action throughput of broadcast_requesting while w_1 changes (r_1=0.2)

w_1	Population		Action throughput
	Master	Device	
0.03	53.2	84.7	2.54
0.04	40.6	80.6	3.22
0.05	29.1	76.9	3.84
0.06	21.3	74.3	4.26
0.10	21.3	74.3	4.26

According to the above analysis, the action throughput of broadcast_requesting can be drawn as a surface (Fig. 15).

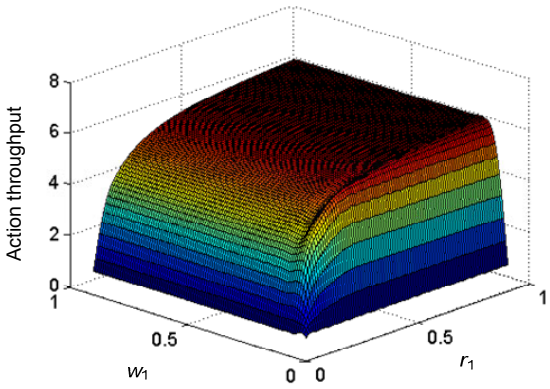

Fig. 15 The action throughput of broadcast_requesting

Second, while r_2 and w_3 are set to a series of different values, the action throughput of the return slides on the surface is shown in Fig. 16. It is easy to obtain the conclusion that the action throughput of return increases with the growth of r_2 and w_3. Furthermore, the growth trend will gradually slow down while r_2 and w_3 become too large. The reason is that the time interval between return sensing results cannot be infinitely reduced and will be stopped at a proper range.

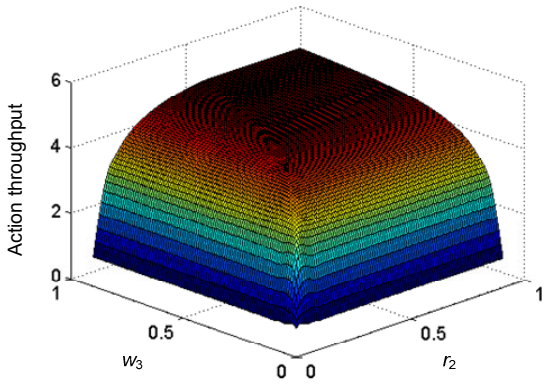

Fig. 16 The action throughput of return

4.5 Comparison with existing methods

To demonstrate the efficiency of our method, we compare it with some existing methods. It needs almost the same time to compute the metrics for different methods with steady-state probability distribution or approximation steady-state probability distribution, but the time used to obtain steady-state probabilities varies greatly. So, primary attention is paid to comparing the time used to solve the model by different approaches including Monte-Carlo simulation, fluid-flow approximation, and the Markov chains before and after simplification.

Taking the model shown in Fig. 4 as an example, it is necessary to compare the computing time with those of other methods under different initial populations of components. In this case, the number of derivations of Master is k_1=12, the number of derivations of Device is k_2=3, and the initial populations of Master and Device are shown as M and N, respectively. So, the number of states of the Markov chains method before simplification is $12^M \times 3^N$. After simplification, the upper bound of the state space of the Markov chains method is $C_{N+k_1-1}^{k_1} C_{M+k_2-1}^{k_2}$. In addition, the Markov chains method and Monte-Carlo simulation method are tested using the eclipse PEPA plug-in tool developed by the University of Edinburgh

(Tribastone, 2007). For the simulation method, the number of transient steps is set to 10 000, the confidence level is 0.99, and all other parameters use the default settings. The experiment was executed in a PC with a 4-core 2.4 GHz CPU, 4 GB RAM, and the results are shown in Table 4.

In Table 4, it is apparent that while M and N increase, the Markov chains method suffers from a serious problem of state-space explosion even after simplification. At the same time, the simulation method takes too long a time to obtain the solution while M and N get larger. Yet, note that there are perhaps more than a million nodes in the MCC systems, and apparently the simulation method does not meet the requirements. Thereafter, the fluid-flow approximation method which costs less time is a good choice.

We can draw the same conclusion while studying the comparison by solving the model in Fig. 5. Compared with the Markov chain or simulation-based methods, it is easier to analyze the availability of MCC by a fluid-flow approximation method.

5 Related works

Nowadays, there has been much research on the availability of cloud and some results can be directly used in the area of MCC. However, we should be aware that the service availability of MCC is much different from that of cloud computing, because of the inherited characteristics of the mobile. Only a few of the current results of service availability are fit for MCC. With the wide application of MCC, the requirements of service availability of MCC systems are increasing.

Since the research on MCC is currently at an early stage, SLA is treated as an important way to guarantee the service availability of MCC (Undheim *et al.*, 2011). Qi and Gani (2012) presented a review on the characteristics, recent research, and future research trends of MCC, and discussed the open problems of SLA and how it should be used for availability. Wu *et al.* (2012) proposed a series of admission control and scheduling algorithms for SaaS providers to effectively use public cloud resources to maximize profit by using a function of SLA. In these studies, the items of SLA have been used to estimate the availability of MCC, but the lack of rapid analysis methods hinders their practical use. In addition, before the target system is deployed, it is hard to predict the availability for SLA.

Using the sampling method to assess the metric of availability is another common approach. The sampling method usually has advantages of being simple, and timeliness, but we must first have a representative test system. In Singh and Kumar (2012), six typical cloud systems from Amazon, Microsoft, Google, and other companies were observed to obtain the metrics of availability on different aspects, and the results were compared through a hexagonal floor plan. Widjajarto *et al.* (2012) believed that the IaaS (Infrastructure-as-a-Service) systems provide resources including compute power, storage, network bandwidth, and powers, so they treated these resources as the metrics of availability and set up an availability analysis model. For a real test system, however, different load levels, different deployments, and other random factors bring large impacts on the measured metrics. Just like SLA, this method is unavailable before the target system is deployed.

Table 4 The comparison of computing time

Parameter	Number of components		Computing time (ms)			
	Markov chains before simplification	Markov chains after simplification	Markov chains before simplification	Markov chains after simplification	Monte-Carlo simulation	Fluid-flow approximation
$M=N=3$	6104	546	10 438	328	3562	31.4
$M=N=10$	3.65×10^{15}	940 576	–	–	8688	31.4
$M=N=100$	4.27×10^{155}	2.43×10^{18}	–	–	51 219	31.4
$M=N=200$	1.82×10^{211}	1.44×10^{22}	–	–	100 938	31.4
$M=N=1000$	2.01×10^{1556}	1.34×10^{31}	–	–	506 281	31.4

'–' means it is beyond the machine's computing power

Apart from the above two approaches, some existing availability evaluation methods used in cloud or MCC are usually based on the state space from a theoretical point of view, for example, the Markov method used in Chilwan *et al.* (2011) to analyze the availability of cloud, the Petri net used in Longo *et al.* (2011) to analyze the availability of IaaS. However, these methods always suffer from the problem of state-space explosion. Even though many methods have been proposed in recent years to deal with state-space explosion, such as state-space decomposition, fixed-point iteration, and state-space aggregation (Ever *et al.*, 2013), it is not easy to meet the demands for computing service availability. Because these methods have to scan the whole state space, the computing time used is an exponential function of the number of all components in the system.

Fluid-flow approximation, also known as 'mean-field approximation', proposed by Hillston (2005), avoids the problem of state-space explosion by converting PEPA into ODEs. The technology arouses a great deal of interest in many fields. In Tribastone *et al.* (2012a) and Castiglione *et al.* (2014), it has been used to deal with state-space explosion of the Petri net and process algebra. Recently, Hayden *et al.* (2012) presented a method for computing global passage time using the technology of mean-field approximation, and our study is partly inspired by the work of Hayden.

6 Conclusions

We propose a service availability analysis method for MCC based on fluid-flow approximation. Some critical metrics of service availability are analyzed, including the response time of service, the minimum sensing time of devices, the minimum number of nodes chosen, and the action throughput. This provides a basis for further studies on the availability of MCC.

Compared to existing methods, the major improvements of our work are:

1. The proposed method is based on a formal language, instead of SLA or measuring a real-world system. Thus, we can analyze the service availability of MCC even before a target system is deployed. Meanwhile, it avoids a variety of random interferences from different real systems under testing.

2. The fluid-flow approximation approach is used to convert the PEPA model into ODEs, avoiding the problem of state-space explosion which is widely existing in current models, such as the Markov chains or Petri net.

3. By studying the core metrics of the service availability of MCC, the impacts of parameters and the initial conditions are analyzed by means of the passage time. Based on these results, the service availability of MCC can be easily estimated.

In this study, only some representative metrics of availability are considered. In the future, more metrics will be studied.

References

Bradley, J.T., Gilmore, S.T., Hillston, J., 2008. Analysing distributed Internet worm attacks using continuous state-space approximation of process algebra models. *J. Comput. Syst. Sci.*, **74**(6):1013-1032. [doi:10.1016/j.jcss.2007.07.005]

Castiglione, A., Gribaudo, M., Iacono, M., *et al.*, 2014. Exploiting mean field analysis to model performances of big data architectures. *Fut. Gener. Comput. Syst.*, **37**:203-211. [doi:10.1016/j.future.2013.07.016]

Chilwan, A., Undheim, A., Heegaard, P.E., 2011. Effects of dynamic cloud cluster load on differentiated service availability. 21st Int. Conf. on Computer Communications and Networks, p.1-6. [doi:10.1109/ICCCN.2012.6289310]

Dinh, H.T., Lee, C., Niyato, D., *et al.*, 2013. A survey of mobile cloud computing: architecture, applications, and approaches. *Wirel. Commun. Mob. Comput.*, **13**(18):1587-1611. [doi:10.1002/wcm.1203]

Ever, E., Gemikonakli, O., Kocyigit, A., *et al.*, 2013. A hybrid approach to minimize state space explosion problem for the solution of two stage tandem queues. *J. Netw. Comput. Appl.*, **36**(2):908-926. [doi:10.1016/j.jnca.2012.10.006]

Fernando, N., Loke, S.W., Rahayu, W., 2013. Mobile cloud computing: a survey. *Fut. Gener. Comput. Syst.*, **29**(1):84-106. [doi:10.1016/j.future.2012.05.023]

Fourneau, J.M., Kloul, L., Valois, F., 2002. Performance modelling of hierarchical cellular networks using PEPA. *Perform. Eval.*, **50**(2-3):83-99. [doi:10.1016/S0166-5316(02)00101-3]

Gens, F., Adam, M., Bradshaw, D., *et al.*, 2013. Worldwide and Regional Public IT Cloud Services 2013–2017 Forecast. IDC Market Analysis #242464.

Hayden, R.A., Bradley, J.T., 2008. Fluid semantics for passive stochastic process algebra cooperation. Proc. 3rd Int. ICST Conf. on Performance Evaluation Methodologies and Tools, p.1-10. [doi:10.4108/ICST.VALUETOOLS2008.4329]

Hayden, R.A., Stefanek, A., Bradley, J.T., 2012. Fluid computation of passage-time distributions in large Markov models. *Theor. Comput. Sci.*, **413**(1):106-141. [doi:10.1016/j.tcs.2011.07.017]

Hillston, J., 2005. Fluid flow approximation of PEPA models. Proc. 2nd Int. Conf. on Quantitative Evaluation of Systems, p.33-42. [doi:10.1109/QEST.2005.12]

Huang, Y., Kintala, C., Kolettis, N., *et al.*, 1995. Software rejuvenation: analysis, module and applications. 25th Int. Symp. on Fault-Tolerant Computing, p.381-390. [doi:10.1109/FTCS.1995.466961]

Huerta-Canepa, G., Lee, D., 2010. A virtual cloud computing provider for mobile devices. Proc. 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, Article 6, p.1-5. [doi:10.1145/1810931.1810937]

Longo, F., Ghosh, R., Naik, K., *et al.*, 2011. A scalable availability model for Infrastructure-as-a-Service cloud. IEEE/IFIP 41st Int. Conf. on Dependable Systems & Networks, p.335-346. [doi:10.1109/DSN. 2011.5958247]

Ou, S., Yang, K., Zhang, J., 2007. An effective offloading middleware for pervasive services on mobile devices. *Perv. Mob. Comput.*, **3**(4):362-385. [doi:10.1016/j.pmcj.2007.04.004]

Qi, H., Gani, A., 2012. Research on mobile cloud computing: Review, trend and perspectives. 2nd Int. Conf. on Digital Information and Communication Technology and Its Applications, p.195-202. [doi:10.1109/DICTAP.2012.6215350]

Rishabh, S., Sanjay, K., Munesh, C.T., 2013. Mobile cloud computing: a needed shift from cloud to mobile cloud. 5th Int. Conf. on Computational Intelligence and Communication Networks, p.536-539. [doi:10.1109/CICN.2013.116]

Satyanarayanan, M., 2011. Mobile computing: the next decade. *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, **15**(2):2-10. [doi:10.1145/2016598.2016600]

Singh, G., Kumar, R., 2012. Availability metrics for cloud vibrant behaviour with benchmarks influence on diverse facets. *Int. J. Softw. Eng. Appl.*, **3**(1):101-115. [doi:10.5121/ijsea.2012.3108]

Tribastone, M., 2007. The Pepa Plug-in Project. 4th Int. Conf. on the Quantitative Evaluation of Systems, p.53-54. [doi:10.1109/QEST.2007.34]

Tribastone, M., Gilmore, S., Hillston, J., 2012a. Scalable differential analysis of process algebra models. *IEEE Trans. Softw. Eng.*, **38**(1):205-219. [doi:10.1109/TSE.2010.82]

Tribastone, M., Ding, J., Gilmore, S., *et al.*, 2012b. Fluid rewards for a stochastic process algebra. *IEEE Trans. Softw. Eng.*, **38**(4):861-874. [doi:10.1109/TSE.2011.81]

Undheim, A., Chilwan, A., Heegaard, P., *et al.*, 2011. Differentiated availability in cloud computing SLAs. 12th IEEE/ACM Int. Conf. on Grid Computing, p.129-136. [doi:10.1109/Grid.2011.25]

Widjajarto, A., Supangkat, S.H., Gondokaryono, Y.S., *et al.*, 2012. Cloud computing reference model: the modelling of service availability based on application profile and resource allocation. Int. Conf. on Cloud Computing and Social Networking, p.1-4.

Wu, L., Garg, S.K., Buyya, R., 2012. SLA-based admission control for a Software-as-a-Service provider in cloud computing environments. *J. Comput. Syst. Sci.*, **78**(5):1280-1299. [doi:10.1016/j.jcss.2011.12.014]