A Cooperative Scheduling Scheme of Local Cloud and Internet Cloud for Delay-Aware Mobile Cloud Computing

Tianchu Zhao, Sheng Zhou, Xueying Guo, Yun Zhao, Zhisheng Niu
Tsinghua National Laboratory for Information Science and Technology
Department of Electronic Engineering, Tsinghua University, Beijing 100084, China
Email: zhaotc13@mails.tsinghua.edu.cn, sheng.zhou@tsinghua.edu.cn, guo-xy11@mails.tsinghua.edu.cn
zhaoyun12@mails.tsinghua.edu.cn, niuzhs@tsinghua.edu.cn

***Abstract*—With the proliferation of mobile applications, Mobile Cloud Computing (MCC) has been proposed to help mobile devices save energy and improve computation performance. To further improve the quality of service (QoS) of MCC, cloud servers can be deployed locally so that the latency is decreased. However, the computational resource of the local cloud is generally limited. In this paper, we design a threshold-based policy to improve the QoS of MCC by cooperation of the local cloud and Internet cloud resources, which takes the advantages of low latency of the local cloud and abundant computational resources of the Internet cloud simultaneously. This policy also applies a priority queue in terms of delay requirements of applications. The optimal thresholds depending on the traffic load is obtained via a proposed algorithm. Numerical results show that the QoS can be greatly enhanced with the assistance of Internet cloud when the local cloud is overloaded. Better QoS is achieved if the local cloud orders tasks according to their delay requirements, where delay-sensitive applications are executed ahead of delay-tolerant applications. Moreover, the optimal thresholds of the policy have a sound impact on the QoS of the system.**

I. Introduction

The amount of mobile applications increased dramatically in recent years. By April 2015, Android users have accessed to more than 1.5 million applications [1]. This trend enhances the Quality of Service (QoS) of mobile devices, but the energy consumption is also increased. In fact, the plethora of applications caused heavy energy consumption, which significantly reduces the battery life of smart phones. Remote execution is a possible way to help smart phones save energy. By offloading energy-intensive tasks to resource-rich servers, battery life of mobile devices can be significantly improved [2]. Based on Mobile Cloud Computing (MCC), some platforms are designed, e.g., MAUI [3], CloneCloud [4].

Although remote execution is very prominent in terms of energy saving, it brings challenges to guarantee latency. Delay is a very important QoS requirement from mobile users [5]. However, if an application is offloaded to a remote centralized cloud server, the delay requirement can hardly be satisfied because of the long transmission delay over the Internet [6] [7]. Cloudlet [8] is proposed to deploy some local cloud servers, so that delay requirement can be met. Some specific architectures focusing on technological details are designed then, such as FemtoCloud [9], CONCERT [10]. In the proposed architecture, each local cloud serves mobile users of several nearby cells, which indicates that local cloud should be deployed densely with a large number. Our earlier work [11] studied and analysed how much computational resources need to be deployed in a cloud so that they can be used efficiently. It is concluded that if computational resources exceeds a threshold, the extra resources only provide marginal gain. Based on the analysis, computational resources of each local cloud should be deployed reasonably so as to balance the cost and QoS.

Thus, although local cloud is beneficial in terms of transmission delay, its computational resources is relatively limited. An architecture to associate cloudlets is proposed in [12], which takes the advantage of cloudlets cooperation to overcome computational resource limitation of a single cloudlet. Yet computational resource of cloudlets still has a limitation, so that system performance might degrade when the traffic load is high. Remote cloud has sufficient computational resources, and it can cooperate with the local cloud to achieve better QoS. Load sharing between the local cloud and remote cloud is studied in [13], which is optimized in terms of average response time and energy consumption. But each application has a delay requirement bound, and it is not practical to evaluate the performance of this kind of traffic by average delay.

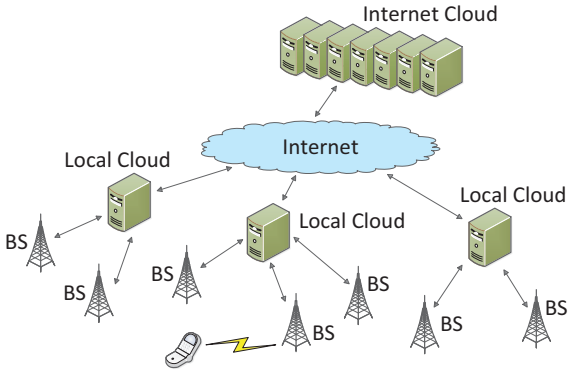

Fig. 1. A cellular network with local cloud and Internet cloud.

As application data needs to be transmitted to remote cloud by Internet, we define remote cloud as Internet cloud. The transmission delay of remote execution can be quite long and the delay jitter is generally large. One simple intuition is that, delay-sensitive applications should be executed in the local cloud, while delay-tolerant tasks can be offloaded to Internet cloud when traffic load is heavy. By the cooperation of local cloud and Internet cloud, computational resources can be used efficiently and QoS requirements can be satisfied.

We illustrate our idea in Fig. 1. Resource-constraint local cloud is near to mobile users, while resource-abundant Internet cloud is remotely located. Mobile users access the local cloud through wireless communication and fronthaul transmission, while Internet transmission is in addition to them if users access the Internet cloud. As local cloud has limited computational resources, some arriving tasks might need to wait longer to be served. To enhance the QoS, we design a scheduling policy so that delay requirements of more users are satisfied. The policy cooperatively schedules the resources in the local cloud and Internet cloud. When the traffic load of the local cloud is above a certain threshold, delay-tolerant applications have to be offloaded to the Internet cloud in order to leave more local computational resources for delay-sensitive applications. To further enhance the QoS, We model the local cloud as a priority queue system. For delay-sensitive applications, they are labeled with higher priority and will be executed ahead of delay-tolerant applications.

The rest of the paper is organized as follows. Section II introduces the system model and gives the problem formulation. Section III proposes the scheduling policy and analyses its performance. Section IV shows numerical results to evaluate the proposed policy. The paper is concluded in Section V.

II. System Model and Problem Formulation

The system model is shown in Fig. 2. Mobile users offload their applications to cloud servers. The data is firstly transmitted to a scheduler which is located in the local cloud. The scheduler decides whether to execute the application in the local cloud or send it to the Internet cloud. The application is then executed in one of the clouds. As soon as the execution is completed, the result will be sent back to the scheduler, and it is finally fed back to mobile users. In this model, wireless transmission and fronthaul transmission between users and the local cloud is needed no matter where to execute the application, whose delay is considered as a small constant τ.

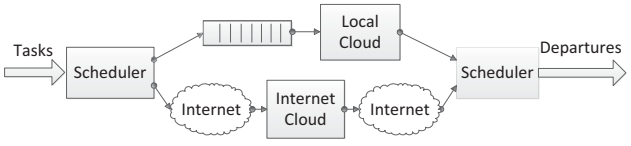

Fig. 2. System model.

Each application offloaded from mobile devices is considered as an arriving task, and each task has a delay constraint. The task is successfully completed if it is executed within the constraint. We design a resource allocation policy to optimize the success probability.

A. Tasks

Assume that there are N types of tasks with different delay requirements. Tasks of type i arrive at the scheduler in a Poisson process with rate λ_i, and they are executed by either the local cloud or the Internet cloud. Each of them has a system delay constraint $T_i (i = 1, ..., N)$ which is the delay requirement minus τ. These tasks request exponentially distributed service time with parameter μ. Rank the priority of these tasks according to delay requirements, and a delay requirement vector $\boldsymbol{T} = (T_1, T_2, ..., T_N)$ is given:

$$T_1 \leq T_2 \leq ... \leq T_N \tag{1}$$

B. Local Cloud

Assume that there are C virtual machines working in the local cloud. Each virtual machine can be seen as a single server. In the queuing system, we assign different priorities to tasks of different delay requirements, where a task of smaller delay requirement has a higher priority. The system forms a nonpreemptive priority queue, where tasks being executed will not be interrupted when a higher priority task comes. Meanwhile, the system has a finite buffer for each type of tasks. Accordingly, the local cloud is modeled as an $M/M/C$ system with modified preemptive priorities. In our model, the total delay is composed of two parts, which includes the queuing delay and execution delay. The local cloud model is shown in Fig. 3.

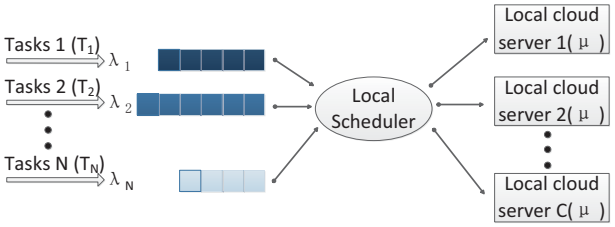

Fig. 3. Local Cloud model.

C. Internet Cloud

The mean delay of Internet transmission is assumed to be long and so does the delay jitter. Thus, Internet cloud is by no means a good choice in terms of delay requirement. However, if the number of coming tasks exceeds the service capacity of the local cloud, Internet cloud should be used to help improve the probability that the delay requirement is met. As the Internet cloud is abundant in computational resources, we assume that the execution delay can be ignored compared with the transmission delay.

Some related works model Internet transmission delay, and we adopt the model of reference [14]. They propose that $\varphi_1(t)$

is the router delay distribution and $\varphi_2(t)$ is the queuing delay distribution, the Internet transmission delay distribution is

$$\varphi(t) = p\varphi_1(t) + q\varphi_1(t) * \varphi_2(t) \tag{2}$$

D. Optimization Objective

The optimization objective is the probability that an arriving task is successfully executed within its delay constraint T_i. The constraint of the system is the limited computational resource of the local cloud. The maximum number of servers can be used in the local cloud is C. The objective is $P_{\text{success}}(t < T_i|C)$, where t indicates the time consumption. We design a scheduling policy to optimize the objective by deciding when and where to execute the arrival tasks.

III. Policy Designment and Performance Optimization

Assume that the arrival rates and service rate of all types of tasks are given, which are $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_N)$ and μ separately. Define the vector $\boldsymbol{S} = (s, l_1, l_2, ..., l_N)$ as the state of the queuing system. The first parameter s denotes the number of busy servers in the local cloud, and $l_i (i = 1, 2, ..., N)$ denotes the number of tasks of priority i waiting in the queue.

We design a threshold based policy to cooperate the local cloud and Internet cloud, which schedules tasks according to their delay requirements. The policy is shown as follows.

Priority-based Cooperation policy: If there is at least one empty server, a task is executed in the local cloud as soon as it arrives. Otherwise, the arrived task of priority i waits in the queue of its own type. If a task departures the system, the empty server will execute a waiting task of the highest priority if any. The priority-based sub-queues are illustrated in Fig. 3 with different colors. For tasks of priority i or higher, a buffer threshold B_i is set to contain them. If the buffer B_i is full, the coming task of priority i or higher is offloaded to the Internet cloud. The buffer thresholds vector $\boldsymbol{B} = (B_1, ..., B_N)$ is derived accordingly.

The intuition of the policy designment is explained as follows. Firstly, tasks of higher priorities have shorter delay requirements. Tasks of lower priorities could wait in the queue until higher-priority tasks have been executed. Secondly, the sojourn time distribution is determined by queue length. By optimizing threshold B_i, the success probability of tasks can be enhanced. Finally, if a priority-i task is the last one in the queue and the queue length equals to the threshold, the policy do not permit a higher-priority task to enter the queue. Otherwise, the priority-i task may suffer from low success probability because of a burst of higher-priority traffic.

To further evaluate the performance of our policy, we design some classical policies for comparison, which are shown as follows.

Local Cloud policy: The tasks are executed only in the local cloud. The system is $M/M/C$ with preemptive priority.

Greedy policy: The coming task chooses the better one between the local cloud and Internet cloud so that it will have a higher success probability.

FCFS-based Cooperation policy: Local cloud is a $M/M/C$ system with First Come First Serve (FCFS) queue. The coming task is offloaded to either the local cloud or the Internet cloud by comparing current queue length with a threshold.

Non-buffer policy: If all local cloud servers are being used, the coming task will be offloaded to the Internet cloud.

A. Stationary Distribution

The queuing system is a N-dimension Markov chain. We can get stationary distribution by formulating and solving global balance equation. Define $L_i = \sum_{j=1}^{i} l_j$ and $\Lambda_i = \sum_{j=1}^{i} \lambda_j$, $\rho_i = \frac{\Lambda_i}{\mu}$. For $L_N = 0$,

$$\begin{aligned}&(\Lambda_N + s\mu)p(s, 0, 0, ..., 0) = \\ &(s+1)\mu p(s+1, 0, 0, ..., 0) + \Lambda_N p(s-1, 0, 0, ..., 0)\end{aligned} \tag{3}$$

For $0 < L_i < B_i$,

$$\begin{aligned}&(\Lambda_N + C\mu)p(C, l_1, l_2, ..., l_N) = \\ &\sum_{j=1}^{N} \lambda_j p(C, l_1, ..., l_j - 1, ..., l_N) \\ &+ \sum_{j=1}^{M} C\mu p(C, l_1, ..., l_j + 1, ..., l_N)\end{aligned} \tag{4}$$

Here, M is the type of tasks of the highest priority in the queue, and the queuing system state is $(C, 0, ..., 0, l_M, l_{M+1}, ..., l_N)$.

For the maximum i satisfying $L_i = B_i$,

$$\begin{aligned}&(\Lambda_N - \Lambda_i + C\mu)p(C, l_1, l_2, ..., l_N) = \\ &\sum_{j=1}^{N} \lambda_j p(C, l_1, ..., l_j - 1, ..., l_N)\end{aligned} \tag{5}$$

This is a N-dimension Markov chain, and it has only one stationary distribution.

Proof: States $(s, 0, ..., 0)$ and states $(C, 0, ..., 0)$ communicate with each other, which is denoted as $(s, 0, ..., 0) \leftrightarrow (C, 0, ..., 0)$. The states also have the following relations.

$$(C, l_1, ..., l_i, ..., l_N) \leftrightarrow (C, 0, l_2, ..., l_i, ..., l_N) \tag{6}$$

$$(C, 0, ..., 0, l_i, ..., l_N) \leftrightarrow (C, 0, , ..., 0, l_{i+1}, ..., l_N) \tag{7}$$

Thus, all states communicate with each other, which indicates that the Markov chain is irreducible. The Markov chain has a stationary distribution and no other stationary distribution exists [15].

If N=2, the 2-dimension Markov chain is shown in Fig. 4. The states (i) which are below the dashed line represent the number of busy servers, where queue is empty. The states (l_i, l_j) which are above the dashed line represent queue lengths of different types of tasks, where all servers are busy. The states (l_i, l_j) whose $l_i = B_1$ or $l_i + l_j = B_2$ indicate that the buffer is full, and arriving tasks will be offloaded to the Internet cloud.

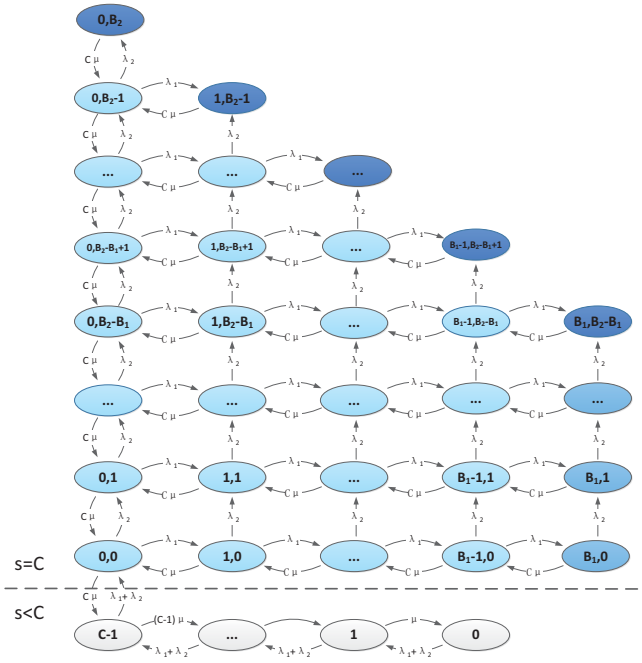

Fig. 4. Markov chain for 2-priorities system.

B. Sojourn Time Distribution

System state is $\boldsymbol{S} = (s, l_1, ..., l_N)$. If $s \leq C$, a task is served as soon as it arrives at the queuing system. The sojourn time equals to the service time, which is exponentially distributed. The pdf of sojourn time is:

$$p_{\text{st}}(t|\boldsymbol{S}) = \mu e^{\mu t} \quad (t \geq 0) \tag{8}$$

If $s > C$, all servers are being used when a task arrives at the queuing system. The task has to wait in the queue or served by the Internet cloud. If the task is served by the local cloud, it needs to wait in the queue until it can be served. The sojourn time consists of waiting time and service time.

Assume that the priority of the arriving task is i. The distribution of waiting time for the task $w(t)$ is $L_i + 1$ fold convolution of $f(t)$ which is the busy period of a C-server system serving higher-than-i priority tasks [16]. The probability density function $f(t)$ and its Laplace-Stieltjes transform are as follow [17].

$$f(t) = \frac{1}{t\sqrt{\rho_{i-1}}} e^{-(\Lambda_{i-1}+\mu)t} I_1(2t\sqrt{\Lambda_{i-1}\mu}) \quad (t \geq 0) \tag{9}$$

$$\bar{F}(s) = (s + \Lambda_{i-1} + C\mu - \sqrt{(s + \Lambda_{i-1} + C\mu)^2 - 4c\mu\Lambda_{i-1}})/2\Lambda_{i-1} \tag{10}$$

I_1 is a modified Bessel function of the first kind. Thus, the distribution of waiting time $w(t)$ and its Laplace-Stieltjes transform is derived.

$$w(t|\boldsymbol{S}) = f(t) * f(t) * ... * f(t) \tag{11}$$

$$\bar{W}(s|\boldsymbol{S}) = \bar{F}(s)^{L_i+1} \tag{12}$$

The sojourn time distribution $p_{\text{st}}(t|\boldsymbol{S})$ is the convolution of waiting time and service time.

$$p_{\text{st}}(t|\boldsymbol{S}) = w(t|\boldsymbol{S}) * \mu e^{\mu t} \tag{13}$$

Given the state $\boldsymbol{S}$ of the system, success probability is

$$P_{\text{success}}(t \leq T_i|\boldsymbol{S}) = \int_0^{T_i} p_{\text{st}}(t|\boldsymbol{S})dt \tag{14}$$

If the task is served by the Internet cloud, the sojourn time distribution is modeled as an empirical distribution $P_1(t)$ which is given in (2).

C. Success Probability

Assume that $\boldsymbol{\lambda}, \mu$ and $\boldsymbol{B}$ are given, the success probability of priority-i tasks is calculated as:

$$\begin{aligned} P_{\text{success}}(i|\boldsymbol{\lambda}, \mu, \boldsymbol{B}) &= P(t \leq T_i|\boldsymbol{\lambda}, \mu, \boldsymbol{B}) = \\ &\sum_{L_j < B_j} P_{\text{st}}(t \leq T_i|\boldsymbol{S}) p(s, l_1, l_2, ..., l_N) \\ &+ \sum_{L_j = B_j} P_1(t \leq T_i) p(s, l_1, l_2, ..., l_N) \end{aligned} \tag{15}$$

Note that the $P_{\text{st}}(t \leq T_i|\boldsymbol{S})$ in equation (15) is related to $\boldsymbol{\lambda}, \mu$ and $\boldsymbol{B}$, which is given in equation (14).

The total success probability is

$$P_{\text{success}}(\boldsymbol{\lambda}, \mu, \boldsymbol{B}) = \frac{\sum_{i=1}^{N} \lambda_i P_{\text{success}}(i|\boldsymbol{\lambda}, \mu, \boldsymbol{B})}{\Lambda_N} \tag{16}$$

D. Local Optimal Thresholds

Search algorithm can be used to get the optimal thresholds vector $(B_1, ..., B_N)$, so that the success probability is maximized. But the complexity of search algorithm might be quite high. Here, we give a low-complexity recursive algorithm to get the local optimal thresholds, which is shown in the Algorithm 1. Firstly, make $(B_1, ..., B_N) = (0, ..., 0)$. Secondly, continually increase B_N by 1 until P_{success} begin to decrease, and a local optimal B_N is derived given that $(B_1, ..., B_{N-1}) = (0, ..., 0)$. Next, make $(B_0, ..., B_{i-1}) = (0, ..., 0)$. Increase B_i by 1 each time to get optimal $(B_{i+1}, ..., B_N)$, and stop adding B_i until P_{success} decreases. Repeat the previous step to get the buffer thresholds $(B_1, ..., B_N)$. This algorithm gives local optimal thresholds, while search algorithm is optimal globally. In the simulation scenarios, numerical results show that the thresholds of our algorithm equal to the thresholds derived by search algorithm.

IV. Numerical Results

We evaluate the proposed priority-based cooperation policy by comparing it with other policies stated previously. In the evaluation, two types of tasks are considered, which are delay-sensitive tasks and delay-tolerant tasks separately. We assume the parameters of the system as follows. Delay requirement of

Algorithm 1 Find local optimal thresholds

Input: $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_N), \mu$
Output: $\boldsymbol{B} = (B_1, ..., B_N)$

$\boldsymbol{B} \leftarrow (0, ..., 0)$
$\boldsymbol{B} \leftarrow$ FINDOPTIMALTHRESHOLD$(1, N, \boldsymbol{B})$

procedure FINDOPTIMALTHRESHOLD(i, N, $\boldsymbol{B}$)
  for $k \leftarrow i$ to N **do**
    $B_k \leftarrow B_{i-1}$
  end for
  if $i = N$ **then**
    $P_{\text{Success1}} \leftarrow P_{\text{success}}(\boldsymbol{\lambda}, \mu, \boldsymbol{B})$
    $P_{\text{Success2}} \leftarrow P_{\text{Success1}}$
    while $P_{\text{Success1}} \leq P_{\text{Success2}}$ **do**
      $B_N \leftarrow B_N + 1$
      $P_{\text{Success1}} \leftarrow P_{\text{Success2}}$
      $P_{\text{Success2}} \leftarrow P_{\text{success}}(\boldsymbol{\lambda}, \mu, \boldsymbol{B})$
    end while
    $B_N \leftarrow B_N - 1$
  else
    $\boldsymbol{B} \leftarrow$ FINDOPTIMALTHRESHOLD$(i + 1, N, \boldsymbol{B})$
    $P_{\text{Success1}} \leftarrow P_{\text{success}}(\boldsymbol{\lambda}, \mu, \boldsymbol{B})$
    $P_{\text{Success2}} \leftarrow P_{\text{Success1}}$
    while $P_{\text{Success1}} \leq P_{\text{Success2}}$ **do**
      $B_i \leftarrow B_i + 1$
      $\boldsymbol{B} \leftarrow$ FINDOPTIMALTHRESHOLD$(i+1, N, \boldsymbol{B})$
      $P_{\text{Success1}} \leftarrow P_{\text{Success2}}$
      $P_{\text{Success2}} \leftarrow P_{\text{success}}(\boldsymbol{\lambda}, \mu, \boldsymbol{B})$
    end while
    $B_i \leftarrow B_i - 1$
    $\boldsymbol{B} \leftarrow$ FINDOPTIMALTHRESHOLD$(i + 1, N, \boldsymbol{B})$
  end if
  return $\boldsymbol{B}$
end procedure

delay-sensitive tasks is 50 milliseconds, and delay requirement of delay-tolerant tasks is 300 milliseconds. The Internet delay is modeled in (2), whose mean delay is 200 milliseconds. For the local cloud server, its mean service time is 10 milliseconds.

Fig. 5 shows the comparison between priority-based cooperation policy and local cloud policy. When traffic load is low, the local cloud has enough computational resources to execute arriving tasks and most users can complete their tasks within delay requirements. However, the success probability decreases dramatically with the increasing of arrival rate. In fact, most users have to wait in the queue when traffic load is heavy, which leads to poor QoS. In this case, cooperation of the local cloud and Internet cloud is quite necessary. By offloading delay-tolerant tasks to Internet cloud, much more mobile users can have their applications completed successfully. In our model, more than 20% success probability of tasks can be enhanced by cooperation of the local cloud and Internet cloud when traffic load is heavy.

Fig. 6 shows the comparison between priority policy and non-priority policies. A single user can achieve higher QoS

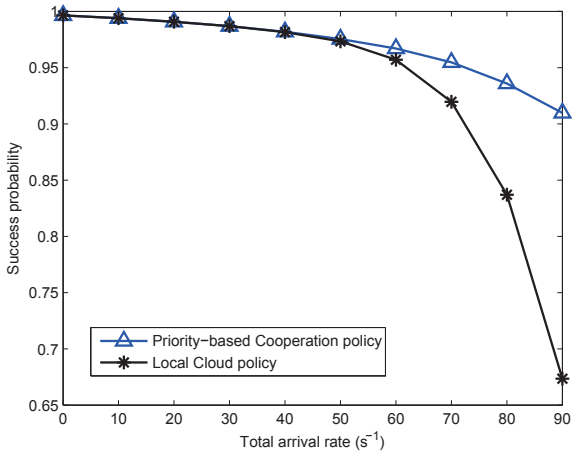

Fig. 5. Success probability vs. Arrival rate for Priority-based Cooperation policy and Local Cloud policy.

by greedy policy which maximizes his success probability. However, optimization of a user becomes a burden for the system, because it results in longer mean waiting time. In fact, local optimum is by no means global optimum. The scheduling policy needs to be designed globally so that higher success probability for total users can be achieved. For the FCFS-based cooperation policy, it fully utilizes the computational resources of the local cloud and the Internet cloud and its performance is quite good. But higher QoS can be realized by considering priorities of tasks. Results shows that the priority policy is better than the FCFS policy to a certain extent. In our model, it results in a 5% success probability improvement if the policy considers priorities of tasks. Non-buffer policy only makes decisions according to the state of servers. It fails to utilize the buffer to make future plans and further improve the QoS, which results in a bad QoS performance.

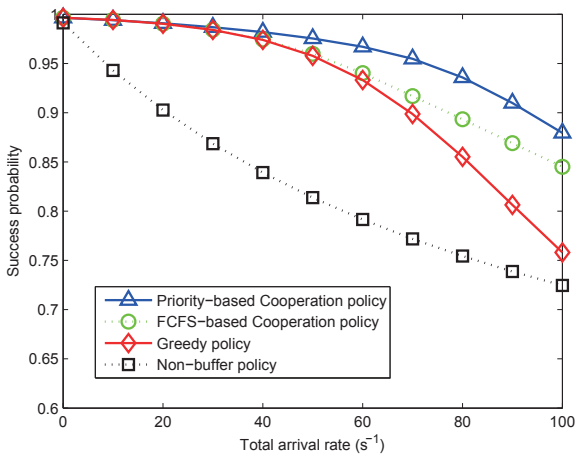

Fig. 6. Success probability vs. Arrival rate for Priority-based Cooperation policy and Non-Priority Cooperation policies.

Fig. 7 gives the optimal thresholds of our proposed policy.

When traffic load is low, threshold B_2 is a small value. In this condition, extra buffer is not needed and a small threshold will achieve the optimal success probability. With the increase of arrival rate, a larger buffer threshold is essential to hold more tasks in the local cloud. However, when traffic load is heavy, the buffer threshold should decrease, because the queue will always be full and long queue length leads to large waiting time. Threshold B_1 decreases with the increase of arrival rate, which leads to waiting time reduction of all types of tasks.

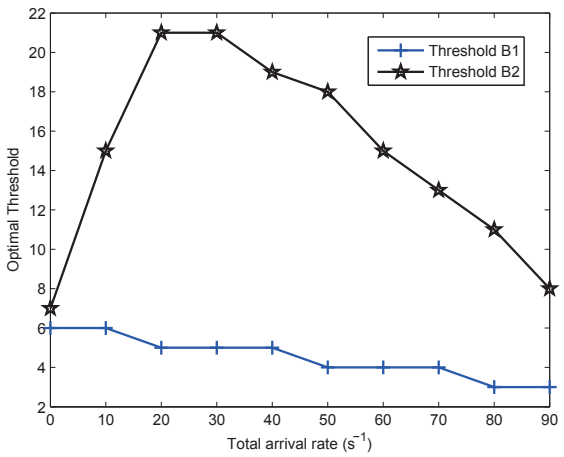

Fig. 7. Optimal Thresholds vs. Arrival rate.

V. Conclusion

In this article, we have improved the QoS of MCC users by designing a scheduling scheme to realize the cooperation between the local cloud and the Internet cloud. We firstly classify applications according to their delay requirements, and give higher priority to applications with shorter delay requirements. Then, we design a threshold-based policy to cooperatively scheduling the local cloud and the Internet cloud, so that the QoS is dramatically improved. By optimizing the thresholds, probability that tasks can be executed within their delay requirements is maximized. We further give an recursive algorithm to get the optimal thresholds with low computation complexity. Numerical results reveal that: 1) Limited computational resources of the local cloud greatly influences the QoS when the traffic load is high, and Internet cloud is needed to improve QoS. By cooperation of the local cloud and Internet cloud, probability that a task is completed within its delay requirement can be improved by 20%. 2) The QoS can be further improved by 5% via a priority scheme which executes delay-sensitive tasks ahead of delay-tolerant tasks. 3) Optimal buffer thresholds are tightly related to the traffic load. As the traffic load increases, a larger buffer threshold is needed to hold more tasks. But thresholds should decrease to guarantee a small waiting time when traffic load is heavy.

Acknowledgment

This work is sponsored in part by the National Basic Research Program of China (973 Program: No. 2012CB316001), the National Science Foundation of China (NSFC) under grant No. 61201191, No. 61322111, No. 61321061, No. 61401250, and No. 61461136004, and Hitachi Ltd.

References

[1] [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps

[2] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, 2013.

[3] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. Int. Conf. Mobile Syst.*, 2010, pp. 49–62.

[4] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proc. 2011 European Conference on Computer Systems*, 2011, pp. 301–314.

[5] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *IEEE Computer*, no. 4, pp. 51–56, 2010.

[6] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay, "Variability in tcp round-trip times," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003, pp. 279–284.

[7] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, pp. 40–45, 2006.

[8] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[9] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Communicating while computing: Distributed mobile cloud computing over 5g heterogeneous networks," *IEEE Signal Processing Magazine*, vol. 31, no. 6, pp. 45–55, 2014.

[10] J. Liu, T. Zhao, S. Zhou, Y. Cheng, and Z. Niu, "Concert: a cloud-based architecture for next-generation cellular systems," *IEEE Wireless Communications*, vol. 21, no. 6, pp. 14–22, 2014.

[11] J. Liu, S. Zhou, J. Gong, Z. Niu, and S. Xu, "On the statistical multiplexing gain of virtual base station pools," in *IEEE GlobeCom14, Austin, USA*, 2014.

[12] J. Rawadi, H. Artail, and H. Safa, "Providing local cloud services to mobile devices with inter-cloudlet communication," in *IEEE Mediterranean Electrotechnical Conference (MELECON)*, 2014.

[13] E. Gelenbe, R. Lent, and M. Douratsos, "Choosing a local or remote cloud," in *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, 2012.

[14] G. Hooghiemstra and P. Van Mieghem, "Delay distributions on fixed internet paths," Delft University of Technology, Tech. Rep., 2001.

[15] S. M.Ross, *Applied Probability Models with Optimization Applications*, 1970.

[16] R. H. Davis, "Waiting-time distribution of a multi-server, priority queuing system," *Operations Research*, vol. 14, no. 1, pp. 133–136, 1966.

[17] I. Adan, *Course QUE: Queueing Theory, Fall 2003: The M/M/1 system*, 2012.