SCIENCE CHINA
Information Sciences

• **Supplementary File** •

Multisite computation offloading in dynamic mobile cloud environments

Xiaomin JIN[1,2*], Yuanan LIU[1,2], Wenhao FAN[1,2], Fan WU[1,2] & Bihua TANG[1,2]

[1]*School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing* 100876*, China;*
[2]*Beijing Key Laboratory of Work Safety Intelligent Monitoring, Beijing University of Posts and Telecommunications, Beijing* 100876*, China*

Appendix A Application model

The application model is the abstract of mobile applications and is composed of many components (e.g., classes, functions). An application can be represented by a graph $G = (V, E)$ illustrated in Figure A1. A vertex v ($v \in V$) is modeled as a 3-tuple $\{w_v, l_v, o_v\}$. w_v is the amount of component c_v's instructions or CPU cycles which can be obtained by application analysis. l_v represents whether component c_v is offloadable or not. Some components of an application cannot be offloaded because they need to operate local hardware resources. For example, some components need to operate sensors of the mobile device. In this case, these components must be executed in the mobile device. These slash-filled circles in Figure A1 denote the unoffloadable components and we assume that the first component is always executed in mobile devices. o_v is the index of component c_v and indicates the execution order of component c_v. For instance, component c_2 will be executed after the completion of component c_0 and component c_1, and needs the output data of these two components. An edge $e = (u, v)$ ($u, v \in V$) represents the interactive relationship between component c_u and component c_v and its weight $d_{u,v}$ denotes the amount of interactive data. If two components have no interactive relationship, the edge weight will be set to zero. In some cases, the last component c_N has output data which will be passed to component c_0 before the application's completion.

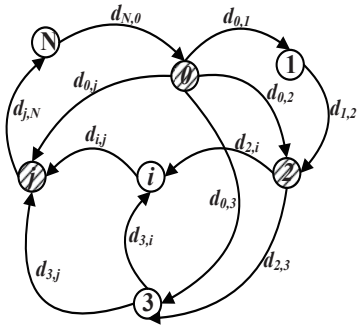

Figure A1 Application model.

Appendix B Pseudocode of our algorithm

Algorithm B1 illustrates the pseudocode of our runtime application repartitioning algorithm. When a component of an application is about to be executed, a request is sent to our algorithm for obtaining the offloading strategy. After receiving an offloading request, our algorithm returns the offloading strategy of that component. Then the component is executed according to the offloading strategy returned.

* Corresponding author (email: jxm@bupt.edu.cn)

Algorithm B1 Our runtime application repartitioning algorithm

calculate chromosome length cl_0, *energy cost* $E_l(G)$, *time cost* $T_l(G)$;
initialize memory $M(0)$ *and population* $P(0)$;
while *simulting* **do**
  if *receive the strategy request* **then**
    $//B_P(h)$ *is the best chromosome*
    $sendStrategy(X, B_P(h))$;
    $//env$ *represents the environmental parameters*
    $updateMemory(M(h), F_h^1, env, B_P(h))$;
  end if
  if *environment changes* **then**
    recalculate F_h^2 *and update envrionmental parameters*;
  end if
  genetic operations;
  $//memory-based immigrants$
  $//B_M(h)$ *is the best memory*
  $P_I(h) = mutateBestMemory(B_M(h), r_i \times M, pr_m^i)$;
  $evaluate(P_I(h), F_h^1, F_h^2)$;
  $//P_n(h)$ *is the new population after genetic operations*
  $replace(P_n(h), P_I(h))$;
  $P(h+1) = P_n(h)$;
end while

Appendix C Objective function in dynamic mobile cloud environments

Figure C1 illustrates an example of the objective function in dynamic mobile cloud environments.

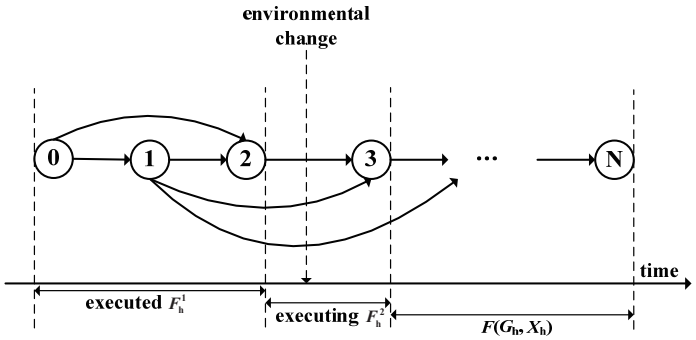

Figure C1 Example of objective function in dynamic mobile cloud environments.

Appendix D Memory and element

The memory and element are illustrated in Figure D1.

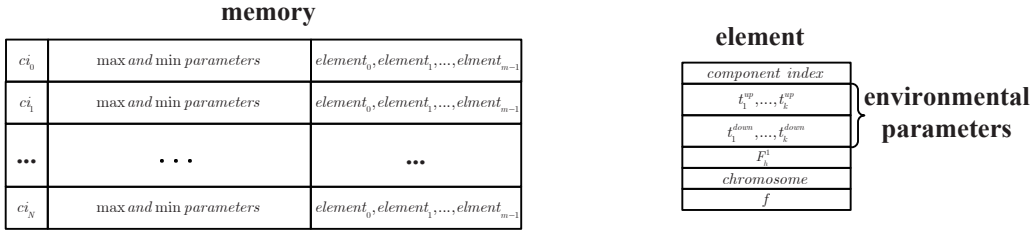

Figure D1 Memory and element.

Appendix E Mobility Model

In this letter, we propose a mobility model to simulate the dynamic mobile cloud environments. Different from models in [1] and [2], the trajectory generated by our mobility model is unknown and users keep moving during applications' life cycle.

We use a 3-tuple $\{s, \Delta t, \gamma\}$ to represent a user's mobility and the mobility model is illustrated in Figure E1. s represents the area where the user locates. To simplify the network environment, network parameters in a square are stable. In square s_j, network parameters are modeled as a set $t_j = \{(t_i^{up}, t_i^{down}) | i = 1, ..., k\}$ whose element represents the uplink and downlink throughput between cloud server cs_i and the mobile device. Δt is the time that the user take to cross square s. A large Δt means that the user crosses square s with a slow speed. On the contrary, a small Δt means that the user crosses square s with a fast speed. γ is the probability that the user will still stay in the same square after Δt. $(1-\gamma)/l$ is the probability that the user will cross to an adjacent square where l is the number of a square's adjacent squares. Users in each square have eight movement directions which are shown at the right part of Figure E1 and the number of directions decreases in marginal squares. For example, users in square S_{12} can move to S_6, S_7, S_8, S_{11}, S_{13}, S_{16}, S_{17} and S_{18}; users in square S_0 can move to S_1, S_5 and S_6. A user's trajectory is modeled as a vector $tr = \{(s_1, \Delta t_1), (s_2, \Delta t_2), ..., (s_n, \Delta t_n)\}$ and a trajectory $(S_{20} \rightarrow S_9)$ example is given in Figure E1.

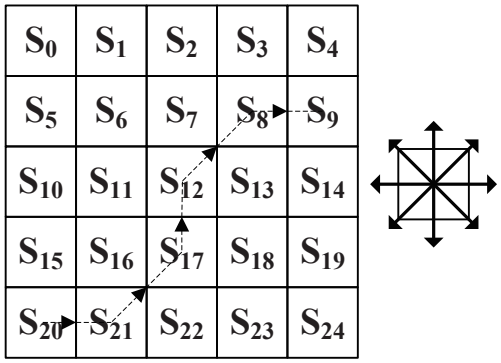

Figure E1 Mobility model.

Appendix F Experiment

In this section, we conduct other different experiments to evaluate our algorithm and the using of concurrent multipath transfer (CMT). Experiments run on a PC with Intel Core i5 CPU (4 cores, 3.3GHz, 4.0G RAM). Parameters of our algorithm are set as: $M = 50$, $r_i = 0.2$, $pr_m^i = 0.01$, $s_{th} = 0.6$. We first illustrate the numerical simulation platform where these experiments are conducted. We then conduct experiments to evaluate our algorithm and explore the using of CMT with our algorithm. If there is no additional description, $w_e = w_t = 0.5$. Six runtime application repartitioning algorithms are used in this part.

ILP1: Algorithm based on integer linear programming (ILP). It resolves the ILP problem when detecting environmental changes and uses local execution strategy as default.

ILP2: Algorithm based on ILP. It resolves the ILP problem periodically and uses local execution strategy as default.

SGA: Algorithm based on standard genetic algorithm (SGA) which reruns when detecting environmental changes.

AGA: Algorithm based on adaptive genetic algorithm (AGA) which reruns when detecting environmental changes.

MISGA: Algorithm based on memory-based immigrants SGA.

MIAGA: Our algorithm based on memory-based immigrants adaptive genetic algorithm.

Appendix F.1 Numerical simulation platform

The numerical simulation platform is illustrated in Figure F1. The application execution simulation module simulates the application's execution. It sends a request to the algorithm module to get the offloading strategy of the component which is about to be executed. After receiving the offloading strategy request, the algorithm module will return the offloading strategy of the component to the application execution simulation module. The random trajectory generation module generates user trajectory according to the mobility model which is introduced in Appendix E. The environment monitoring module monitors the environmental change and notifies the algorithm module and the application execution simulation module.

Appendix F.2 Evaluation of our runtime algorithm

Figure F2 shows the weighted total cost of five different applications. Applications' number of components is 20, 40, 60, 80 and 100, respectively. Results of ILP-based algorithms (ILP1 and ILP2) are close to that of GA-based algorithms when they are used in Application1. Application1's number of components is relatively small for ILP-based algorithms so that they take less time to make offloading decisions, which enables ILP-based algorithms to adapt to environmental changes timely. However, the performance of ILP-based algorithms becomes bad with the increase in the number of components. Running time ILP takes to solve a problem is exponential with the number of variables so that ILP-based algorithms take much more time to partition an application when the application's number of components is large. Long running time makes them unable to adapt to environmental changes and provide wrong offloading strategies. The performance of

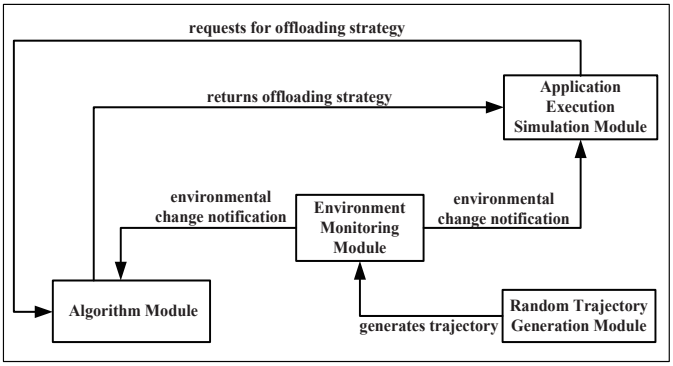

Figure F1 Numerical simulation platform.

GA-based algorithms is better than that of ILP-based algorithms and the advantage of MIAGA is more obvious when the number of components is large. Results of AGA-based algorithms (AGA and MIAGA) are better than that of SGA-based algorithms (SGA and MISGA). The reason is that SGA's convergence rate is slow and SGA is easy to fall into the local optimal solution. Although MISGA has a high convergence rate with the help of the memory, it is also easy to fall into the local optimal solution because its memory stores the SGA's solutions which are not optimal. Compared with SGA, AGA can get better solutions by costing little extra time because it just needs a simple calculation of crossover and mutation probabilities. Results of AGA are a little better than that of MIAGA in Application1 and Application2. This is because that these two applications' number of components is relatively small for AGA so that AGA can find optimal solutions quickly but memory-based immigrants may affect solutions of MIAGA because it needs to increase the diversity to adapt to environmental changes. This problem can be solved by reducing r_i and MIAGA's advantages are highlighted in the large scale, poor computing resources and high users' speed scenarios, which is illustrated in following experiments.

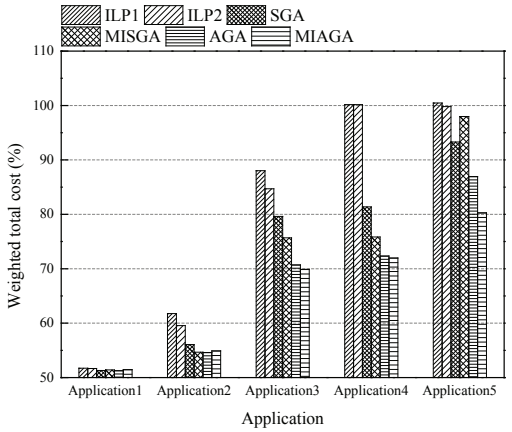

Figure F2 Weighted total cost of different applications.

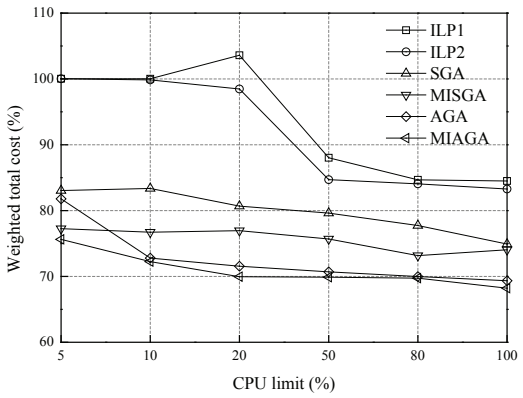

Figure F3 Weighted total cost with different CPU limit.

Application repartitioning algorithms may be executed in different types of equipments (e.g., mobile devices, PCs or cloud servers) and computing resources may become less even in the same equipment because of other programs' execution. Some algorithms may perform well in powerful equipments but not in weak equipments. In this experiment, we limit the CPU usage of runtime application repartitioning algorithms with a Linux command *cpulimit* to simulate scenarios with different types of equipments. Figure F3 shows the weighted total cost of Application3 with different CPU limit. Results of ILP-based algorithms are close to 100% when CPU limit is 5% and 10%. The reason is that they cost much time to get the offloading strategy with small CPU limit and use the default local execution strategy. Results of MIAGA are better than that of other algorithms. MIAGA performs well and its results are relatively stabler (from 75.63% to 68.22%) than that of AGA (from 81.79% to 69.35%) with different CPU limit. When the CPU limit is 5%, the result of MIAGA (75.63%) is much better than that of AGA (81.79%). This because that MIAGA can get better solutions quickly with the help of the historical offloading strategies.

Figure F4 illustrates the weighted total cost with different speed of the user. In this experiment, we use the same trajectory (moving path is same) with different speed. λ is the coefficient of the time that the user takes to cross a square. A larger λ represents that the user takes more time to cross a square, which means the user's speed is slow. Slow speed brings less environmental changes. Results of ILP-based algorithms decrease (ILP1: from 94.24% to 85.31%; ILP2: from 92.69% to 85.90%) gradually with the decrease of the user's speed. MIAGA still performs well with different speed. When

the user's speed is fast ($\lambda = 0.5$), the result of MIAGA (70.34%) is obviously smaller than that of AGA (73.72%).

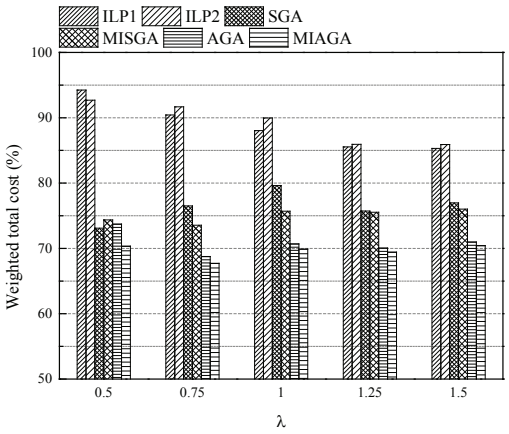

Figure F4 Weighted total cost with different speed.

Appendix F.3 Computation offloading using CMT

In this part, we explore the using of CMT with our algorithm in computation offloading of MCC. We use two physical interfaces (Wi-Fi and LTE) in these experiments. The mobile device can use Wi-Fi and LTE to transfer data simultaneously by using CMT. MIAGA has two functions in the computation offloading of using CMT. One is to determine whether to offload a component or not. Another is to determine which physical interface should be used to transfer data according to the optimization objective.

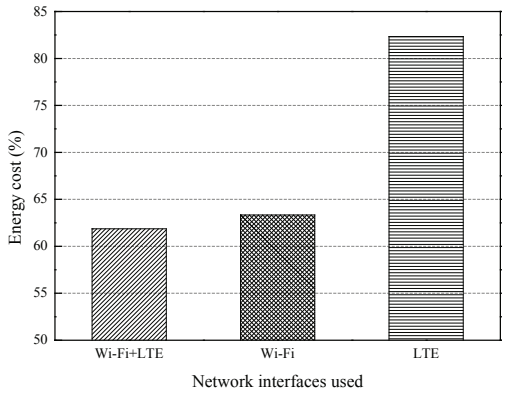

Figure F5 Energy cost with different interfaces.

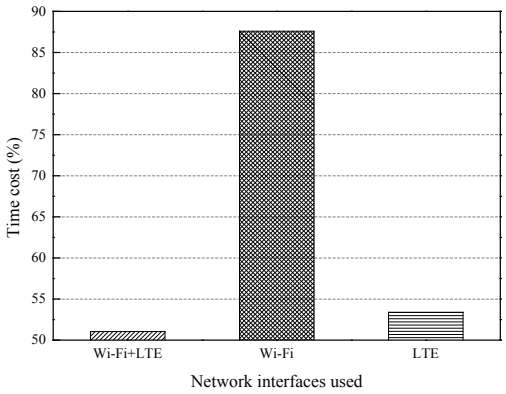

Figure F6 Time cost with different interfaces.

Figure F5 illustrates the energy cost ($w_e = 1$, $w_t = 0$) with different interfaces. It can be seen that CMT has the smallest energy cost (61.87%) and LTE has the largest energy cost (82.33%). In this letter, we use formula $p = \alpha t + \beta$ [3] to model the power of wireless network interfaces. In this formula, p represents the uplink (downlink) power, t represents the uplink (downlink) throughput, α is a constant determined by uplink or downlink, and β is a constant determined by the interface type. By using this power model, the energy cost brought by data transmission can be expressed as $e = p\frac{d}{t} = d(\frac{\beta}{t} + \alpha)$ where d is the amount of data needed to be transferred. β_{LTE} (1288.04mW) is a large value which represents the tail power consumed by the connection between the mobile device and base station. Large tail power makes LTE much less energy efficient than Wi-Fi in most cases. That is the reason why the energy cost of only using Wi-Fi is smaller than that of only using LTE. However, LTE usually has higher uplink and downlink throughput than Wi-Fi, which makes it possible for LTE to have less energy cost in some cases. This also provides an opportunity to optimize energy consumption by using CMT. In this experiment, we use energy cost as the optimization objective. MIAGA will determine whether to offload a component or not and select the optimal interface to transfer data according to this optimization objective, which gives CMT the best result.

Figure F6 shows the time cost ($w_e = 0$, $w_t = 1$) with different interfaces. LTE has higher throughput than Wi-Fi in most cases, which makes the result of LTE smaller than that of Wi-Fi. It can be observed that CMT brings greatest time savings (51.04%) and the performance of Wi-Fi (87.59%) is worst. In this experiment, we use time cost as the optimization

objective. According to this objective, MIAGA will select the optimal interface to transfer data to save as much time as possible.

References

1 Huang D, Wang P, Niyato D. A dynamic offloading algorithm for mobile computing. IEEE Trans Wirel Commun, 2012, 11: 1991-1995
2 Deng S G, Huang L T, Taheri J, et al. Computation offloading for service workflow in mobile cloud computing. IEEE Trans Parallel Distrib Syst, 2015, 26: 3317-3329
3 Huang J X, Qian F, Gerber A, et al. A close examination of performance and power characteristics of 4G LTE networks. In: Proceedings of the 10th international conference on Mobile systems, applications, and services, Low Wood Bay, 2012. 225-238