(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 April 2017 (27.04.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/067586 A1

(51) International Patent Classification: *G06F 9/50* (2006.01)

(21) International Application Number: PCT/EP2015/074319

(22) International Filing Date: 21 October 2015 (21.10.2015)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: DEUTSCHE TELEKOM AG [DE/DE]; Friedrich-Ebert-Allee 140, 53113 Bonn (DE).

(72) Inventors: **HUI, Pan**; Hong Kong University of Science and Technology, 1 University Road Apartment 29, HKUST, Clear Water Bay, Hong Kong (CN). **HUANG, Di**; Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong (CN). **YANG, Ji**; Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong (CN). **PEYLO, Christoph**; Nordweg 14, 49401 Damme (DE).

(74) Agent: **RAINER, Viktor**; VOSSIUS & PARTNER Patentanwälte Rechtsanwälte mbB, Siebertstraße 3, 81675 München (DE).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

(54) Title: METHOD AND SYSTEM FOR CODE OFFLOADING IN MOBILE COMPUTING

(57) Abstract: A hybrid offloading network system for offloading computational tasks, comprises a server cloud that is adapted, when receiving an offloading task, to execute the offloading task; an ambient cloud comprising a plurality of smart devices, wherein each smart device is adapted to deliver, as an offloader, its offloading task and attributes of the offloading task, and/or is adapted, when receiving an offloading task, to execute, as an offloadee, the offloading task. A smart router manages information on currently available computational resources both in the server cloud and the ambient cloud. The smart router receives offloading tasks from the ambient cloud, decides which candidate among the server cloud and the offloadee(s) of the ambient cloud, each received offloading task is to be forwarded to, based on the resource information of each candidate, the attributes of the offloading task and/or network bandwidth, and forwards the offloading task to the decided candidate.

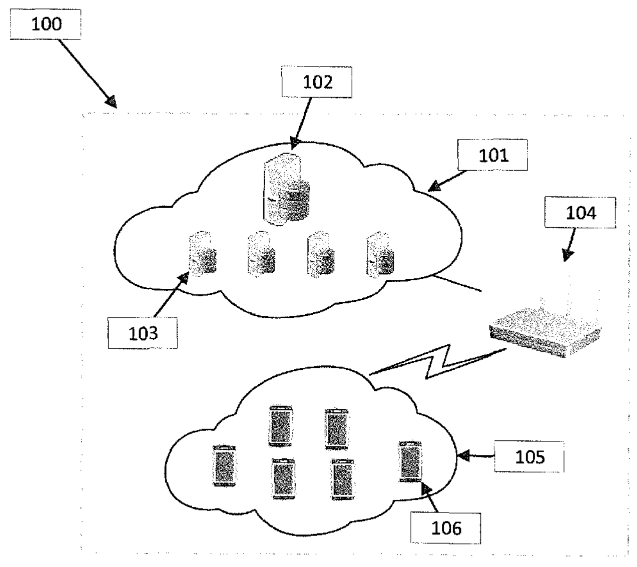

FIG. 1

METHOD AND SYSTEM FOR CODE OFFLOADING IN MOBILE COMPUTING

The present invention is related to building a hybrid code offloading system in mobile computing. This hybrid offloading system is a combination of a cloud comprising one or more servers (also known as server cloud) and an ambient cloud comprising a plurality of smart devices (also known as cirrus cloud or mobile devices cloud).

Mobile computing has been proposed for long with the rapid growth of smart device users. The smart device such as smartphone (e.g. iPhone ^{TM}) is a computational device. Mobile computing involves the interaction of humans, laptops, cloud servers and smart devices. And there are many components in mobile computing such as communication and hardware. Despite the fact that hardware has been upgraded with a tremendous speed, the application execution speed still bothers both the developer and the smart device users.

The emergence of smart devices such as smartphones has inspired the developers to develop more fantastic applications like mobile gaming and mobile augmented reality. Even with the rapid growth of hardware technology, smart devices still cannot meet the demand of these computation-exhausting applications. To cope with such contradiction, researches these years start to focus on the software solutions-based computation from the cloud (i.e. server cloud). Mobile cloud computing (MCC) is proposed to speed up the smart device applications.

Actually there have been presented many initial works on how to solve the execution problem in MCC by computation offloading. The concept of computation offloading comes from cyber foraging [1], [2], which was proposed a decade ago. The cyber foraging systems utilize nearby resource-rich infrastructures like desktops to help mobile phones execute. They believe that public areas are equipped with such infrastructures in the future. Since then a great amount of works start to exploit cloud offloading [3], [5], [8], [9], [10]. MAUI [5] builds the system based on .NET framework to automatically create both the cloud and smartphone versions of code, which greatly benefits the developers. CloneCloud [3], ThinkAir [10], COMET [8] and Cuckoo [9] create VMs to configure smartphones' running environment on cloud servers, where offloadable methods in smartphones can be directly executed. These systems work well when network bandwidth is high. More recently, researches started to work on the device to device (D2D) offloading, where mobile phones offload tasks to other devices when network bandwidth is poor [12], [11]. These two case studies show the potential and feasibility of D2D offloading by measuring the execution capabilities and energy savings.

Some programming frameworks like Honey bee [7], Misco [6] are also proposed to support device to device computing. The more systemic work for device to device offloading was done by Serendipity [14]. Serendipity takes a PNP-block as input and tries to minimize the execution time with task allocation algorithm. However, the link predictability in the algorithm is too strong to implement in real world. COSMOS [13] tries to provide services for offloading users, which minimizes the offloading cost while maintaining performance.

To sum up, these offloading systems can be categorized into two categories: cloud offloading and ambient offloading. In cloud offloading, smart devices (mobile computational devices) can offload tasks to the cloud server, e.g. with 3G or WiFi connections. Smart devices achieve execution speedup and energy saving when they are well connected to the remote cloud. Usually such a system creates various several VMs (virtual machines) for fast execution, which increases a lot of monetary cost. And the offloading performance degrades as the bandwidth goes down. More recent works consider ambient offloading, the other extreme where smart devices have no Internet access. Smart devices leverage the intermittent connection opportunities with nearby devices for offloading. They can offload tasks to other devices when they are connected e.g. with WiFiDirect or Bluetooth. These ambient offloading systems can work as a supplementary solution for offloading when cloud offloading (by means of a server cloud which is built by mainframes) is inaccessible. Ambient offloading, however, suffers from vulnerability since the connections are often broken up due to mobility. The drawbacks in both the remote and ambient cloud hinder the popularization of offloading systems.

[1] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 87–92. ACM, 2002.

[2] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Sim- plifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 272–285. ACM, 2007.

[3] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proc. of ACM EuroSys*, pages 301–314, 2011.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[5] 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. of ACM MobiSys*, pages 49–62, 2010.

[6] A.Dou,V.Kalogeraki,D.Gunopulos,T.Mielikainen,andV.H.Tuulos. Misco: a mapreduce framework

for mobile systems. In *Proc. of ACM PETRA*, page 32, 2010.

[7] N. Fernando, S. W. Loke, and W. Rahayu. Honeybee: A programming framework for mobile crowd computing. In *Proc. of MobiQuitous*, pages 224–236. Springer, 2013.

[8] R.Geambasu,A.A.Levy,T.Kohno,A.Krishnamurthy,andH.M.Levy. Comet: An active distributed key-value store. In *Proc. of USENIX OSDI*, pages 323–336, 2010.

[9] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applica- tions, and Services*, pages 59–79. Springer, 2012.

[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proc. of IEEE INFOCOM*, pages 945–953, 2012.

[11] A.Mtibaa,A.Fahim,K.A.Harras,andM.H.Ammar.Towardsresource sharing in mobile device clouds: Power balancing across mobile devices. In *Proc. of ACM MCC*, pages 51–56, 2013.

[12] C. Shi, M. H. Ammar, E. W. Zegura, and M. Naik. Computing in cirrus clouds: the challenge of intermittent connectivity. In *Proc. of ACM MCC*, pages 23–28, 2012.

[13] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura. Cosmos: Computation offloading as a service for mobile devices. In *Proc. of ACM MobiHoc*, 2014.

[14] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura. Serendipity: enabling remote computing among intermittently connected mobile devices. In *Proc. of ACM MobiHoc*, pages 145–154, 2012.

[15] L. Zhang and Tiwana, Birjodh et. al: Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of CODES/ISSS*, pages 105–114, 2010.

SUMMARY OF THE INVENTION

The present invention provides a brand new offloading system, which leverages both the cloud and ambient offloading to build a powerful offloading system. The system contains three components: a remote cloud (server cloud), a smart router and an ambient cloud. Remote cloud serves as the basic infrastructure for the offloading systems. It has the most powerful computational resources. The server cloud includes at least a server and storage means. The cloud can be any number of remote servers that are networked to allow a centralized data storage. Most computational exhausting tasks are usually delivered to these places when bandwidth is good. Ambient cloud is formed by the smart devices connected to the smart router. These smart devices serve as masters or slaves. When a smart device is busy with its own tasks, it would be a master. Otherwise it is a slave who is idle and can wait for execution. Smart router is the controller of the offloading systems. When master node decides to offload tasks to the cloud (which will be either a server cloud or other smart device), it will send tasks to the smart router, which decides the executor of this task according to the optimization goal. In the context of the present invention, the router is a computational device (comprising an operating system) and therefore a smart router.

The present invention aims at overcoming the drawbacks and limitations brought by both the cloud and ambient offloading. That is, the present invention is directed towards combining both the cloud and the smart devices together, to build a hybrid offloading architecture to dynamically allocate computational resources.

More specifically, according to an aspect of the present invention, a hybrid offloading network system for offloading computational tasks is provided. The system comprises a first cloud comprising one or more servers and adapted, when receiving an offloading task, to execute the offloading task; and a second cloud comprising a plurality of smart devices. Each smart device is adapted to deliver, as an offloader, its offloading task and attributes of the offloading task, and/or is adapted, when receiving an offloading task, to execute, as an offloadee, the offloading task. A router manages information on currently available computational resources both in the first cloud and the second cloud, receives offloading tasks from the second cloud, decides which candidate among the first cloud and the offloadee or offloadees of the second cloud, each received offloading task is to be forwarded to, based on said resource information of each candidate, said attributes of the offloading task and/or network bandwidth, and forward the offloading task to the decided candidate.

The attributes of the offloading task may preferably include a data size and/or an instruction count

In a preferred embodiment, the router may apply a linear programming to make the decision. The linear programming includes an optimization function to optimize a total cost under the constraints of (i) the network bandwidth and execution time of each offloading task and/or (ii) energy consumption of said offloading task. The total cost is a sum of sub-costs that are each defined per candidate, each sub-cost being a cost that will be incurred if the offloading task is executed by said candidate.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates the architecture of a hybrid offloading system according to an embodiment of the present invention.

FIG. 2 illustrates the software architecture of a cloud shown in FIG. 1.

FIG. 3 illustrates the software architecture of a smart router shown in FIG. 1.

FIG. 4 illustrates the software architecture of a smart device shown in FIG. 1.

FIG. 5 is a flowchart illustrating how to process a task in the smart device.

FIG. 6 is a flowchart illustrating how to process multiple tasks in the smart router.

FIG. 7 is a flowchart illustrating how to train a decision model in the smart device.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Some preferred embodiments will be described with reference to the drawings. For explanation purpose, various specific details are set forth, without departing from the scope of the present invention as claimed.

SYSTEM OVERVIEW

This section mainly explains the system architecture of a hybrid offloading system according to an embodiment shown in FIG. 1. Three different kinds of software are deployed in cloud servers (component 102 and 103), a smart router 104 and smart devices 106.

The cloud server end 101 has two different types: a main server 102 and a plurality of sub-servers 103. The main server 102 comprises a virtual machine manager 201 (FIG. 2). The virtual machine manager 201 responds to requests from the smart router 104, manages creation and destruction of virtual machines 203 (FIG. 2). The virtual machine manager 201 creates virtual machines 203 using resources of the sub-servers 103. Virtual machines 203 are only responsible for execution. These two components (virtual machine manager 201 and virtual machines 203) together form an essential part of the remote cloud 101.

In the smart router end 104, a task scheduler 302 (FIG. 3) allocates the executors of tasks submitted by smart devices as offloaders. The smart router 104 serves to minimize the total cost for renting cloud servers while maintaining the offloading performance.

In the smart device end 106, a master decision maker 403 (FIG. 4) will decide whether to offload or not according to performance need. All of the smart devices that are connected with the smart router 104 form an ambient cloud 105.

The remote cloud 101, the ambient cloud 105, and the smart router 104 form a hybrid offloading system according to the present embodiment.

Referring to FIG. 2, the software stack for the main server 102 will be explained. In addition to the virtual machine manager 201, the main server 102 comprises a request listener 202 for responding to requests from the smart router 104. The identification of each request is verified in this listener 202 to make sure that the request comes from the smart router 104. If the verification is passed, the socket connection between the main server 102 and the smart router 104 can be set up and preserved. And the request listener 202 will respond to all the requests from the smart router 104. The core component in the virtual machine manager 201 is a virtual machine scheduler 201. It has two basic functions: managing all the virtual machines 203 and performing the scheduling for all tasks received from the smart router 104. The scheduling algorithm is greedy but highly efficient, which takes only *O(nlog(n))* time for n submitted tasks. It should be noted that in the context of the present invention, "scheduling" in the virtual machine manager 201 is determining which task is to be sent to which virtual machine, while "scheduling" in the smart router 104 (which will be explained below) is determining which task is to be sent to which executor (i.e. server cloud or offloadee of the ambient cloud).

Referring to FIG. 3, the smart router 104 according to the present embodiment of the hybrid offloading system comprises a recovery machine 301, a task scheduler 302, a network manager 303, and a task queue 304.

When the master decision maker 403 of the smart device 106 decides to offload a task, the smart router 104 will immediately receive it and add it to the task queue 304. The task queue 304 estimates a current arrival rate of a plurality of submitted tasks (i.e. how many tasks are submitted to the queue in one unit time (usually in one second)), which will help the task scheduler 302 for fast and efficient task allocation.

The task scheduler 302 is the core part in this router 104. The smart router 104 works as the proxy for all the smart devices connected to this router. The smart router 104 manages the resources (e.g. number of virtual machines) both in the server cloud 101 and the ambient cloud 105. The network manager 303 is used to measure the network condition (in particular, network bandwidth) with the server cloud 101. As will be explained in detail below, according to current resources of both the server cloud 101 and the ambient cloud 105, attributes of each received offloading task, and/or the network bandwidth notified from the network manager 303, the task scheduler 302 is configured to decide which executor the task is forwarded to.

The recovery machine 301 in the smart router 104 is different in terms of operation from a local recovery machine 406 (FIG. 4) integrated in the smart device 106: the recovery machine 301 of the smart router 104 does not request the local execution of this task, but request the smart router 104 itself to execute the task). It must instruct another executor (another smart device or the cloud) to execute the task.

Referring to FIG 4, the smart device 106 according to the present embodiment of the hybrid offloading system comprises an application tracker 402, a master decision maker 403, an energy model 404, a profiler 405, and a local recovery machine 406.

The application tracker 402 tracks all applications 401 running in the smart device 106 and specifies (identifies) each offloadable method. The application tracker 402 uses a task queue (not shown) to maintain all the offloadable methods submitted by the applications 401. The task queue applies a first-come-first-serve strategy where each offloadable method is equally served. Node who wants to offload tasks is called master node.

Every offloadable method in the task queue should be sent to the master decision maker 403 to decide whether to offload or not the offloadable method. The decision maker 403 also invokes the profiler 405 to obtain hardware parameters (e.g. CPU status such as CPU utilization, CPU frequency, and battery level) and method parameters (i.e., instruction count) before decision.

For this purpose, the smart device 106 may further comprise a (not shown) execution time estimation means for estimating execution time of each offloadable method based on the hardware parameters and the instruction count of the offloadable method by referring to the profiler 405. The decision maker 403 uses the estimated execution time when making the decision. The smart device 106 delivers, when the decision maker 403 decides that the offloadable method is to be offloaded, to deliver the estimated execution time as an attribute of the offloadable method to the smart router 104.

Once the tasks are offloaded to the smart router 104, the local recovery machine 406 is invoked to monitor the remote execution. The local recovery machine 406 backs up each delivered offloading task, monitors a remote execution of the offloading task. For this purpose, the smart router 104 is adapted to monitor a remote execution of each received offloading task in the server cloud 104 or in the ambient cloud 105 and report a monitoring result to the smart device 106. The local recovery machine 406 monitors a remote execution on the basis of the monitoring result from the smart router 104. If errors happen in this system, the local recovery machine 406 will respond immediately to call a local execution of this task.

The energy model (energy consumption estimation means) 404 is the individual part in this system where the energy consumption of offloadable methods is estimated. Specifically, the energy model 404 estimates energy consumption of each offloadable method based on the hardware parameters and the instruction count of the offloadable method by referring to the profiler 405. The estimation is saved in the local database of the smartphone for future use. Using this value (i.e. energy consumption estimation) with respect to the offloadable method, the decision maker 403 may learn to perform a more precise estimation by actually executing the offloadable method within the smart device 106 and compare the actual energy consumption with an estimated value. In this case, the actual execution time may also be compared with an estimated execution time to allow the smart device 106 to perform a more precise estimation of the execution time of a current task.

WORK FLOW

This section mainly discusses how the system operates when a task is generated at a smart device and submitted to this system.

A. Smart device workflow

Reference is made to FIG 5. For application developers, there are generally two programming choices. One way is to do precompiling (step 502), which means that the developers only have to annotate the offloadable method and use precompiling program to convert the original java method to the version that can be posted to the offloading system. Another way is to use the system API (application programming interface) (step 501), which means that the developers have to explicitly post the tasks to the offloading system (step 503) at the point of method invocation. As will be appreciated by the skilled person, any other platform than Java may be used.

Once the task has been posted, the smart device 106 starts the profiling by the profiler 405 (step 504), which are useful for the master decision maker 403 and the energy model 404. Then the master decision maker 403 makes the offloading decision (step 505) according to SVM (support vector machine) model (FIG. 7). The SVM classifier decides whether to offload or not according to the profiling information. After the decision, the smart device starts to send the code to the smart router 104 (step 506). If the code starts execution on an executor selected by the smart router 104, the program will use e.g. the "future" mechanism provided by java to wait for notification of an execution result (step 507). In the end the profiler 405 is stopped and all collected data will be written into a (not shown) local database (storage) of the smart device 104 (step 508).

B. Smart router workflow

Like the application tracker 402 in the smart device 106, the smart router 104 also has a task queue 304 to maintain all the tasks submitted to the router. There will be an individual thread to query and fetch tasks from the queue. As shown in FIG. 6, the smart router 104 will probe available computation resources (step 601) and the current bandwidth (step 602) by sending requests to the connected devices of the ambient cloud 105 and the server cloud 101. With these resources, the router 104 decides the offloading targets with linear programming (step 603). Afterwards each task is sent to the decided offloading target (step 604) and the recovery module 301 is invoked (step 605). Finally the result is sent back through the smart router 104 to the original smart device (step 607) which has offloaded the task.

C. SVM training workflow

Before the usage of the master decision maker 403 in smart devices, the system preferably has to train the parameters (i.e. runtime parameters and profiling information mentioned below) in this model. As shown in FIG. 7, the training process first runs multiple applications to collect the data (step 701). The data collection needs to monitor the runtime parameters like CPU usage and bandwidth. So the training flow will also get the profiling information (e.g. battery usage) and stores these data into the local database of the smart device (step 702 and 703). With enough data entries, the system will use the training algorithm to get the initial decision parameters of SVM (step 704 and 705). However, these initial parameters could have the over-fitting problems if the training process uses only this small partial of data. To overcome this problem, an algorithm may be used to dynamically correct the parameters (step 706). Finally the training process can get the corrected SVM parameters (step 707).

DESIGN DETAILS

The design of this system can be divided into three parts: the smart devices, the smart router and the server cloud. The specific details of these parts will be explained in the following subsections.

A. Smart Device Design Details

Preferably, the design for the smart device has to meet the requirements of both the end users and the application developers. For the application end users, the hybrid offloading system should be efficient and powerful enough to deal with the speedup and the energy problem. Therefore the master decision maker 403 (FIG. 4) of the smart device can make right decisions according to current local hardware status. For the application developers, they mainly focus on their application designs rather than the optimization over the ambient cloud 105. But we have to acknowledge that the application can run smoother if the application developers obey some rule. Therefore we provide two levels of programming for these application designers.

1) Profiler 405: profiler design contains all the information of the hardware parameters, the application parameters and the network. It should be noted that the master decision maker 403 of the smart device 106 is able to decide that tasks should not be offloaded if the network condition is not good (or does not satisfy a predetermined criteria).

As a hardware profiler the profiler 405 may monitor the following things:

- CPU utilization and CPU frequency state
- Screen brightness which ranges from 0 to 16
- WiFi signal strengths indicated by RSSI value
- Power state of the WiFi interfaces

As an application profiler, the profiler 405 mainly monitors the applications in the runtime. It can monitor the following things:

• Execution time of a method

• Number of instructions of a method (i.e. instruction count)

• Input and output parameters of a method (e.g. data type, data size)

• Number of threads invoked by a method

Since we only use WiFi to setup connections, the profiler 405 as a network profiler is simplified by considering:

• RTT of the remote execution servers

• Bandwidth of the remote execution servers

• The connected stream interfaces

2) Energy model 404: Energy model serves two functions in this system: i) providing benchmarks for the master decision maker 403 as to whether to offload the offloadable task and ii) evaluating energy consumption in experiments. In the hybrid offloading system, we take the inspiration from PowerTutor [15], a powerful tool that can have measurement error less than 6.27%. PowerTutor takes the CPU, screen, GPS, WiFi into consideration. We also add the Bluetooth estimation into the PowerTutor to improve the estimation accuracy.

3) Programming rule with the hybrid offloading system: In our design, we propose two different architectures for the application developers to use. In these two proposed architectures, we can both use the Java reflection mechanism to call the method in the remote execution.

Annotation mechanism

This architecture is very friendly to the application developers. Like what has been proposed in previous offloading architectures, the programmers only need to mark the offloadable method before the definition with annotation(@offloadable). The hybrid offloading system invokes the analyzer to generate a runnable version with the offloading services. And the developer's version of code is also added to the remote code version for rewriting and generation of a new apk file. This annotation mechanism has little limitations on the programmers. They just need to specify the code that can be run in other machines. However, this mechanism is not powerful enough to support better optimization like parallel execution.

Post task mechanism

Actually most of the programmers are aware of the code they write. Usually they know how to make the system efficient by parallel method invocation. So in this way, we propose the other programming rules for the developers: to post offloadable tasks to the hybrid offloading system and then fetch the results when necessary. For example, a batch of methods is invoked in a loop. But all the results are collected after for loop. This case can be well dealt in the post task mechanism since all of these methods can be posted to the hybrid offloading system without being blocked by the previous invocation. All results will be preserved in the ResultTicket class and the programmers can fetch the return parameters at the time when the program needs. Compared with the annotation method, this post task mechanism can support better for parallel execution while it takes some troubles for the programmers to adapt.

4) Recovery machine 406: A recovery machine 406 is preferable in the offloading since tasks submitted for remote execution often suffer from failures in terms of network and operating systems. The recovery machine 406 of the smart device (offloader) 106 is needed to specify what occurs and make quick response when failure occurs. To make failure detection easier, we define two exception types: RemoteException and NetworkException. RemoteException designates errors which may occur within remote devices (cloud servers or offloadees in the ambient cloud). If RemoteException happens, the smart device will call a local execution. For this purpose, the smart router 104 monitors a remote execution of each offloading task in the server cloud 101 or the ambient cloud 105 and report a monitoring result to the smart device (offloader). The local recovery machine 406 backs up each delivered offloading task, monitors a remote execution of the offloading task on the basis of the monitoring result from the smart router 104, and calls a local execution of the offloading task in the smart device.

The master decision maker 403 may mark each exceptional remote device. For example, the remote device will be excluded if it continuously throws exceptions 2 times.

NetworkException designates errors which may occur when connection is lost or the socket streams are blocked. In this case, the smart router 104 will reping and reconnect the remote device. The master decision maker 403 of the smart device 106 waits for the remote device's results provided that the remote device and the smart router 104 are re-connected. Otherwise the smart router 104 reports to the smart device 106 that the reconnection fails, and the smart device 106 then calls a local execution in a similar manner as in the case of RemoteException.

B. Smart Router Design Details

The smart router end is actually the local decision center. Besides the basic connection and communications, the smart router 104 may contain a recovery machine 301, which is different in terms of operation from the recovery machine 406 in the smart device end.

The offloading target in the smart router 104 contains two types: the cloud servers and the smart device executors (i.e. offloadees). Cloud servers are usually regarded as the stable computational resources where few errors happen except the API version. But the smart devices often suffer from vulnerability since the smart devices (such as smartphones) sometimes move and disconnect to the smart router 104. The main contributor for the recovery machine 301 in the smart router end is to deal with the abrupt failure brought by smart device executor. In an embodiment, the smart router may make backup for every task in the smart device, putting them in a waiting list. In this case, if any of the executors has some problems for the execution, the smart router itself may start execution immediately.

C. Cloud Server Design

The cloud server provides a general service entrance for all users who want to offload. The java reflection mechanism requires the applications to be compiled and send the apk file for dynamic calling and indexing. And each time before the sending of offloading method, the cloud will check whether the corresponding apk file exists or not. Every apk file is installed with the name of the hash (apk) value. This naming has an advantage which can specify the version changes of the applications. Thus different versions of the same application can be dynamically invoked at the same time.

The virtual machine manager 201 (FIG. 2) serves as two main functions: to manage the virtual machines and to schedule the tasks (i.e. determining which task is to be supplied to which virtual machine for execution). The virtual machine manager 201 finds the daily patterns for the traffic and tries to open corresponding numbers of virtual machines 203. The creation and destruction of virtual machine instances is implemented, for example, by installing the AWS CLI packages. By this way the virtual machine manager 201 can directly create and stop the virtual machines 203.

DECISION AND SCHEDULING

Generally, there are three parts for the offloading decisions and scheduling. These three parts are logically independent but they can cooperate to achieve both the performance improvement and cost minimization. In the smart device end, people are more concentrated on the performance the system achieves. They tend to take the execution speedup or the energy saving as the ultimate goal. For this purpose, we use a support vector machine (SVM) to decide whether each offloadable method should be offloaded or not based on hardware parameters (such as CPU status, battery life) and instruction count of said offloadable method. When the offloadable method is delivered, as an offloading task, to the smart router, it starts the second level decision on which instance to offload the task, preferably by optimizing a total cost under predetermined constraints by applying a linear programming, which will be explained below. The goal in this decision machine is to make balances between the smart device helpers and the cloud virtual machines. If the task scheduler 302 (FIG. 3) of the smart router 104 decides to offload the task to the ambient cloud, the task will be directly offloaded to the selected candidate. If the task is sent to the server cloud, the virtual machine manager 201 (FIG. 2) of the server cloud will schedule the task to balance the load between the virtual machines 203 (i.e. decide the virtual machine to which the task is to be sent). And the virtual machine manager 201 will manage the VM creation and destruction to save the cost (typically, monetary cost).

A. Master Decision Maker 403

When smart device users access the cloud services, they tend to care about two things: computation speedup and energy saving. They usually want to acquire more computational resources for faster execution while they prefer to save energy in extreme cases where the battery level is lower than some level.

To achieve these goals, we design a support vector machine (SVM) based model to make binary decisions. In this model, we train two classifiers respectively: time incentive classifier and energy incentive classifier.

1) SVM Background: SVM [4] has become a fundamental classification algorithm in supervised learning. Unlike probabilistic linear classifier, SVM classifies the training examples with space mapping and decision. Examples with n features are mapped into an n-dimensional space and the SVM is to find one or several hyperplanes to divide these labeled points. So the hyperplane (also the classifier)

is defined as following:

$$f(x) = w^T x + b \quad (1)$$

where x is the feature vector of an example. For an example xi, it belongs to one class if $f(x_i)<0$, otherwise it belongs to the other class. Specific w and b determine a classifier. When there is some x, we calculate f(x) and check if it is less than zero.

2) Feature Selection & Classification: Although there are two classifiers for this application, the parameter selections are similar because of the correlation between energy consumption and execution speedup. Due to space limitation, we only present the feature choice of time incentive classifier.

$$G(T_i) = t_c + t_s + t_e + t_r - t_l \quad (2)$$

As presented in Eq. 2, we use the $G(T_i)$ to denote the loss of remote execution. The remote execution is beneficial only when $G(T_i) < 0$. tc is the connection setup time with the router, t_s is the method sending time, te is the remote execution time, t_r is the time for results return and t_l is the time for local execution. t_c is usually 0 or a certain value that can be directly estimated by the profiler. t_s is determined by the data size(s) and the current bandwidth estimated by the signal strength (*rssi*). t_e is determined by the executor CPU frequency and the number of instructions (*inum*). t_r is estimated by the return parameter size (r) and the future bandwidth which can also estimated as the *rssi*. And the t_l is determined by both *inum* and current CPU utilization (*util*). Finally the feature x in Eq. 1 for time incentive classifier is $x = (t_c, s, rssi, inum, r, util)$. Besides features used in time incentive classifier, the energy incentive classifier also takes current battery level (bat) and the screen brightness (sb) into consideration. Therefore the feature selected for energy classifier is $x = (t_c, s, b, rssi, inum, r, util, bat, sb)$.

In SVM classification model, we need to know whether these classes are linearly separable. In our scenario, it is obvious that the features we select are linearly inseparable. To deal with this we use the Gaussian kernel to make it separable:

$$k(x_1, x_2) = \exp\left(-\frac{\| x_1 - x_2 \|^2}{2\sigma^2}\right) \quad (3)$$

3) Deployment: Our SVM based master decision maker has a training process whose training dataset comes from running test of applications. The training process randomly generates tasks and each task has two copies, one of which runs on the cloud while the other runs in the smart device. The profiler records all the features and labels the data according to the performance of this task. After the testing, the profiler offloads all these data to the cloud to do parameter training. The cloud will return the vector back to the smart device when the training is done.

B. Task Scheduler 302

This section will explain a preferable method of deciding, at the smart router end, the candidate to which each offloading task is to be forwarded. In this method, the smart router 104 makes a balance between smart devices and cloud servers while causing less harm to the smart devices. The smart router 104 minimizes the total cost for the task execution, preferably with linear programming.

1) Problem Formulation: With the decision made by the master decision maker 403 of the smart device 106, the task profiles are sent to the smart router with the goal of time or energy saving. Although these goals are different in the smart device end, these goals are the same to the router because execution speedup for tasks in the smart router end means both the time and energy savings. Therefore we omit the energy concern in the smart router end.

The linear programming includes an optimization function to optimize a total cost under the constraints of (i) the network bandwidth and execution time of each offloading task and/or (ii) energy consumption of said offloading task. Sub-costs are each defined per candidate, each sub-cost being a cost that will be incurred if the offloading task is executed by said candidate. A sum of sub-costs is defined for each offloading task. A total cost is a sum of the sum of sub-costs for multiple tasks.

As one example, let us assume that a task T_i (i.e. i-th task received by the task queue 304) arrives and then the smart router decides which instance (including both the VMs and smart devices) I_k (i.e. k-th execution instance) to execute. We formulate the scheduling problem as:

$$\begin{aligned}
Min \quad & \sum_{i=1}^{n} \sum_{k=1}^{m} C(T_i, I_k) \qquad (4) \\
s.t. \quad & \sum_{k=1}^{m} I_k(T_i) = 1 \\
& M(T_i, I_k) + \frac{D_s(T_i) + D_r(T_i)}{b} < L(T_i) \\
& \frac{E_o(I_k) - E_c(I_k)}{E_o(I_k)} < \delta
\end{aligned}$$

where n and m are the number of tasks and instances respectively. n and m are an integer of 2 or more (as will be explained below, n can take a value of 1). D_s is an original data size (to be sent from the smart device through the smart router to the execution instance via the network), D_r is a result data size (to be sent back from the execution instance to the smart device via the network; the smart router may receive and forward the result, but the result may be directly sent from the execution instance to the smart device) and b is the bandwidth (e.g. in units of kbps). $M(T_i,I_k)$ is an estimated execution time in the chosen execution instance and $L(T_i)$ is a remaining lifetime (i.e. execution time in the offloader which would be required for Ti plus waiting time in the task queue 304 of the router) of the task T_i. The optimization goal of this smart router is to minimize the total cost of the execution of all these tasks submitted to the smart router and preserved in the queue.

The first constraint means that only one copy of method is executed. The second constraint means that the total time of offloading should meet the deadline of T_i. The third constraint, however, is to protect the helpful smart devices from energy hole when it helps others with offloading. Eo here means an original energy level of the executing instance I_k. Ec is a remaining energy level after the execution of that task. We set a threshold δ in order to ensure that a large among of energy will not be consumed when assigning the offloading task to the smart device (offloadee) in the ambient cloud. Apparently, if the executing instance I_k is a virtual machine instance in the cloud, the third constraint is not considered.

Besides the above-mentioned linear programming model, we can have multiple other alternatives to solve the cost optimization problem by using different constraints. For example, in order to make sure that the offloader can get the results from the executing instance before the time constraint, we can make the first constraint like this:

$$\begin{aligned}
Min \quad & \sum_{i=1}^{n} \sum_{k=1}^{m} C(T_i, I_k) \\
s.t. \quad & \sum_{k=1}^{m} I_k(T_i) = 2 \\
& M(T_i, I_k) + \frac{D_s(T_i) + D_r(T_i)}{b} < L(T_i) \\
& \frac{E_o(I_k) - E_c(I_k)}{E_o(I_k)} < \delta
\end{aligned}$$

(4')

In this case, we can guarantee that once one offloadee fails to transfer the results back, the smart device can get the results from the other device.

Furthermore, we can take the reputation of each smart device into consideration. Some of the smart device may be malicious which can return false results. So the linear programming is like this:

$$\begin{aligned}
Min \quad & \sum_{i=1}^{n} \sum_{k=1}^{m} C(T_i, I_k) \\
s.t. \quad & \sum_{k=1}^{m} I_k(T_i) = 1 \\
& R(I_k) > \theta \\
& M(T_i, I_k) + \frac{D_s(T_i) + D_r(T_i)}{b} < L(T_i) \\
& \frac{E_o(I_k) - E_c(I_k)}{E_o(I_k)} < \delta
\end{aligned}$$

(4")

where $R(I_k)$ is the reputation of instance I_k. We set a threshold θ for choosing the instance that has good executing reputation. This reputation value is calculated via evaluating the execution history of that node. For example, if a smart device (offloadee) only returns a result once every 3 times, the reputation can be calculated as 1/3 and compared with a predetermined value such as 0.5.

2) Equilibrium Based Cost Definition: In order to solve the linear programming defined in the previous subsection, we have to define the sub-cost C (T_i, I_k) (i.e. cost of allocating T_i

to I_k) in formula (4). In actual scenarios, if task Ti chooses a smart device to execute, the real cost (i.e. monetary cost) is zero. If task Ti chooses the server to execute, the cost should be based on the quantity of computation the user uses (i.e. charge). But this falls into a greedy cycle where tasks will be allocated to the smart devices first when it meets the satisfaction for task deadline. And helpers cannot get paid after helping execution. In this way, we need to set a total cost or sub-cost to make balance, which can inspire the helpers to help. The baseline for the cost design is to do less harm to the smart devices when they are ready to help. As one example, we set a sub-cost as:

$$C(T_i, I_k) = \begin{cases} \frac{1-E_l}{E_l}(1-avg(w)) \cdot avg(p) \cdot c_i & I_k \in S_1 \\ 2 \cdot avg(w) \cdot avg(p) \cdot c_i & I_k \in S_2 \end{cases} \quad (5)$$

where *avg(w)* and *avg(p)* are an average workload of the virtual machines on the cloud servers (or cloud server in case where there is a single cloud server) and an average charge of the virtual machines on the cloud servers (or cloud server in case where there is a single cloud server) per unit time. They range from 0 to 1. With respect to avg(w), 0 represents the virtual machine being not loaded at all; 1 represents the virtual machine being fully loaded. With respect to avg(p), the system designer can set its unit (e.g. currency like Euro) and range. c_i is an instruction count of T_i. S_1 refers to the smart device, and the first half of Eq. 5 designates a cost in case of a smart device executor. S_2 refers to the cloud server, and the second half of Eq. 5 designates a cost in case of a cloud server executor. E_l designates a remaining energy of the smart device after offloading and ranges from 0 to 1. 0 represents no energy; 1 represents full energy.

($1-E_l/E_l$) means that the cost is inversely proportional to E_l. ($1-E_l/E_l$) is larger than 0. This will be large when E_l is close to 0.

Regarding the first half relating to smart device executor, the multiplication with avg(p) relating to virtual machines on the cloud servers has been made in order to set reasonable cost of smart device executor (first half) so as to be comparable with the cost of server executor (second half). In other words, it will take some cost even if the task is executed on the smart device in order to reach a balance of task allocation. Furthermore, we assume the cost of executing certain task on the server is twice as that on the smart device, not considering other conditions (energy or workload).

Regarding the second half, avg(w) x avg(p) means that the cost is proportional to avg(w) as well as proportional to avg(p).

As is apparent from (1-avg(w)) in the first half for the smart device executor and avg(w) in the second half for the server executor, Eq. 5 provides a comparison between the costs of the smart device and the server so that the user tends to choose the lower one. When avg(w) (server workload) increases, the cost of the server gets higher while the cost of the smart device gets lower. It is possible to generalize what is specifically defined in Eq. 5. If certain task is executed on the server, the cost only depends on the workload of the server and the charge. Higher workload or higher charge will lead to higher cost. However, if the task is executed on the smart device, the cost will also largely depend on the energy level of the smart device. If the energy level is low, it will significantly increase the cost.

The definition in Eq. 5 actually can make balance between the cloud servers and the smart devices. If the cost is set to real cost (i.e. monetary cost), the final optimization algorithm would go into greedy strategy where all tasks that can meet the constraints will be offloaded to the especially powerful smart devices. However, these smart devices cannot get any reward from it and thus will deny cooperation. To avoid this problem, we propose to assign some cost from the task submitters and reward the task executors. The cost, however, also changes according to the server runtime environment. When the server is very busy with task execution, the cost of cloud computing would be more expensive while the cost for the smart devices execution will be lower. This strategy can greatly protect the system from sudden job jams when the cloud server is busy.

As will be appreciated by the skilled person, the total cost to be optimized (minimized) may be differently defined as those defined in formulae (4), (4'), (4'') and (5). For example, the total cost may be defined with respect to each task (i.e. n=1), instead of being defined with respect to multiple tasks in a queue. In this case, the total cost is a sum of sub-costs that are each defined per candidate, and the sub-cost is a cost that will be incurred if the single offloading task is executed by the candidate.

Furthermore, although the linear programming is a preferable method, the smart router may decide which candidate each offloading task is to be forwarded using another method that uses resource of information of each candidate, attributes of the offloading task and/or network bandwidth as criteria. Preferably, a data size and/or an instruction count of each offloading task may be used as such attributes of the offloading task.

C. Offloadee Design

Here steps that may be performed by an offloadee who actually executes a task to further improve our invention will be explained.

After executing an offloaded task, the offloadee may record execution information of the task and/or energy consumption required for the task in its local database. The execution information may include execution time and CPU status such as CPU utilization when executing the task. The execution information and/or the energy consumption (which are referred to as history execution information and history energy consumption, respectively) may be provided to the smarter router in order to assist the smart router in allocating future tasks.

By comparing the history execution information (i.e. actually required time and actual CPU status for executing an offloaded, past task) of an offloadee with its corresponding estimated time of the past task, the smart router can more accurately estimate execution time of the same offloadee as a candidate for a current, different task to be offloaded. Furthermore, by comparing the history energy consumption (i.e. actually consumed energy of an offloaded, past task) of an offloadee with its corresponding estimated energy consumption of the past task, the smart router can more accurately estimate energy consumption of the same offloadee as a candidate for a current, different task to be offloaded.

D. Cloud Optimization

When tasks are submitted to the cloud, the cloud should be able to scale (allocate) almost all the tasks submitted by an application. The initial goal for the cloud optimization is to satisfy the deadline requirement of the application while balancing the load for the servers. Besides that, the cloud server manager should preferably try to minimize the total cost for this application. The cost here refers to the resource used in the cloud or the load thereof.

Load Balancing: In our offloading system, we design an online load balancing algorithm to let a faster execution and lower latency for the tasks submitted by the smart router. Actually, the virtual machine manager can also benefit from this algorithm according to the average workload of the virtual machines on the cloud servers. As has been proved by the approximation theory, the solution to find the optimal scheduling is NP-hard. In order to save the scheduling and allocation time, we design a longest processing time based balancing algorithm.

This algorithm sorts the task queue according to the load first and then greedily submit these tasks to the least loaded servers. The time complexity for this algorithm is only O(*nlogn*) where n is the number of tasks in this queue.

```
1  Q and S are the submitted task queue and servers.
2  while true do
3          S.sortByExeTimeDes(); // sort S by execution time in descending order
4          for i=1 to S.size() do //go through each server
5                  $L_i = 0$ //load of i's server;
6          T(i) =ϕ □ //tasks allocated to server i;
7     end for
8     for j=1 to Q.size() do //go through each task in the task queue
9                  i = serverWithLeastLoad();
10                 T(i)=T(i)+j //allocate task j to server i
11                 $L_i \leftarrow L_i + t_j$
12    end for
13 end
14         return RQueue
```

SYSTEM IMPLEMENTATION

A. Applications

To test the effectiveness of this system, we design four different types of methods and a real application. These four different methods are used to test the performance of offloading in terms of execution speed and energy saving while the real world application is used to demonstrate the overall performance and user experience.

1) Specific methods: The testing methods should cover most of the application types. They must be intensive either in computation, memory or bandwidth. Therefore we design four applications: FaceDetection, N- Queen, QuickSort and Sudoku.

FaceDetection. This method detects how many faces are there in a picture. We directly use the FaceDetector class in Android API, which takes a Bitmap object as the method input. The attributes of this method varies according to the parameter input. If the picture size is large, the FaceDetection method can be both computation and bandwidth intensive.

N-Queen. This method is designed to find the possible solutions for N queens on an $N \times N$ chessboard. We use a brute force method to traverse all the possible cases, which takes $O(N^{2N})$ time complexity. This method is only a representation of computational intensive method.

QuickSort. QuickSort has been regarded as the most efficient and representative sorting algorithm. We design this method to take an unsorted array as the input and return a sorted array. This method is computation and bandwidth intensive if the input array is very large. Besides, the method can also be memory intensive since the method is invoked recursively.

Sudoku. Apart from the previous applications, we also design a Sudoku solver method, which takes a 9×9 matrix as input and returns whether there exists solutions of this matrix. Sudoku therefore can be regarded as a lightweight method.

2) Sharplens Application: Sharplens is an augmented application developed with Google glass ^{TM}. Google glass is used to help the poor visual or blind people to recognize the context in front of them. This application first recognizes the hand in the video taken by the camera embedded in the glass. Then it locates the position of the finger and recognizes the paragraph around the finger. Finally the paragraph is presented in prism or read out by the speaker. In this application, this patent applies both of the two types of programming rules to show the execution efficiency.

Claims

1. A hybrid offloading network system for offloading computational tasks, comprising:

a first cloud (101) comprising one or more servers and adapted, when receiving an offloading task, to execute the offloading task;

a second cloud (105) comprising a plurality of smart devices (106), wherein each smart device is adapted to deliver, as an offloader, its offloading task and attributes of the offloading task, and/or is adapted, when receiving an offloading task, to execute, as an offloadee, the offloading task; and

a router (104) adapted to manage information on currently available computational resources both in the first cloud (101) and the second cloud (105), receive offloading tasks from the second cloud (105), decide which candidate among the first cloud (101) and the offloadee or offloadees of the second cloud (105), each received offloading task is to be forwarded to, based on said resource information of each candidate, said attributes of the offloading task and/or network bandwidth, and forward the offloading task to the decided candidate.

2. The system according to claim 1, wherein said attributes of the offloading task include a data size and/or an instruction count.

3. The system according to claim 1 or 2, wherein the router (104) is adapted to apply a linear programming to make the decision,

the linear programming includes an optimization function to optimize a total cost under the constraints of (i) the network bandwidth and execution time of each offloading task and/or (ii) energy consumption of said offloading task, and

the total cost is a sum of sub-costs that are each defined per candidate, each sub-cost being a cost that will be incurred if the offloading task is executed by said candidate.

4. The system according to claim 3, wherein the resource information of each offloadee includes a battery level, and

the sub-cost for the offloadee is higher as the battery level is lower.

5. The system according to claim 3 or 4, wherein the resource information of each offloadee includes a CPU status such as CPU frequency, and the router (104) is adapted to estimate, as said execution time of each offloading task, execution time that will be taken by each candidate, based on said CPU status and said instruction count.

6. The system according to claim 5, wherein each offloadee is adapted to store, as history execution information, execution information such as execution time in its local database when executing a task (referred to as a past task), and supply said history execution information to the router (104), and

the router (104) is adapted, when estimating said execution time of each offloading task, to estimate execution time that will be taken by said offloadee, based on a comparison between said history execution information and its corresponding estimated time of the past task, in addition to the CPU status and the instruction count.

7. The system according to any one of claims 3 to 6, wherein the router (104) is adapted to estimate, as said energy consumption of each offloading task, energy that will be consumed by each candidate among the offloadee or offloadees, based on said resource information and said instruction count.

8. The system according to claim 7, wherein each offloadee is adapted to store, as history energy consumption, energy consumption in its local database when executing a task (referred to as a past task), and supply said history energy consumption to the router (104), and

the router (104) is adapted, when estimating said energy consumption of each offloading task, to estimate energy that will be consumed by said offloadee, based on a comparison between said history energy consumption and its corresponding estimated energy consumption of the past task, in addition to the resource information and the instruction count.

9. The system according to claim 3 or 4, wherein the router (104) comprises:

a task queue (304) for preserving offloading tasks received from the second cloud (105) in a queue; and a task scheduler (302) for scheduling forwarding of the offloading tasks preserved in the task queue (304),

the task scheduler (302) is adapted to optimize said total cost, said total cost being defined with respect to a predetermined number "n" of the tasks preserved in the queue, "n" being an integer of 2 or more.

10. The system according to any one of claims 2 to 9, wherein each smart device comprises:

an application tracker (402) for tracking all applications (402) running in the smart device (106) and identifying each offloadable method;

a task queue for preserving offloadable methods submitted by the application tracker (402) in a queue;

a profiler (405) for profiling hardware parameters of the smart device (106) and an instruction count of each offloadable method; and

a decision maker (403) for deciding whether to offload each offloadable method as an offloading task by invoking the profiler (405).

11. The system according to claim 10, wherein the smart device (106) further comprises:

an energy consumption estimation means (404) for estimating energy consumption of each offloadable method based on the hardware parameters and the instruction count of said offloadable method by referring to the profiler (405), and

the decision maker (403) is adapted to use the estimated energy consumption when making the decision.

12. The system according to claim 10 or 11, wherein the smart device (106) further comprises:

an execution time estimation means for estimating execution time of each offloadable method based on the hardware parameters and the instruction count of said offloadable method by referring to the profiler (405),

the decision maker (403) is adapted to use the estimated execution time when making the decision, and the smart device (106) is adapted, when the decision maker (403) decides that the offloadable method is to be offloaded, to deliver the estimated execution time as one of said attributes of the offloadable method to the router (104).

13. The system according to any one of claims 1 to 12, wherein the router (104) is adapted to monitor a remote execution of each received offloading task in the first cloud (101) or the second cloud (105) and report a monitoring result to the smart device (106) as an offloader, and

the smart device (106) further comprises:

a local recovery machine (406) for backing up each delivered offloading task, monitoring a remote execution of the offloading task on the basis of the monitoring result from the router (104), and calling a local execution of the offloading task in the smart device when a failure occurs in the hybrid offloading network system.

14. The system according to any one of claims 1 to 13, wherein

the first cloud (101) comprises a main server (102) and a plurality of sub-servers (103),

the main server (102) comprises a virtual machine manager (201) for creating virtual machines (103) using resources of the sub-servers (104), and

the virtual machine manager (201) is adapted, when receiving one or more offloading tasks from the router (104), to cause a virtual machine (103) to execute each of said received offloading tasks while balancing load between the virtual machines (103).

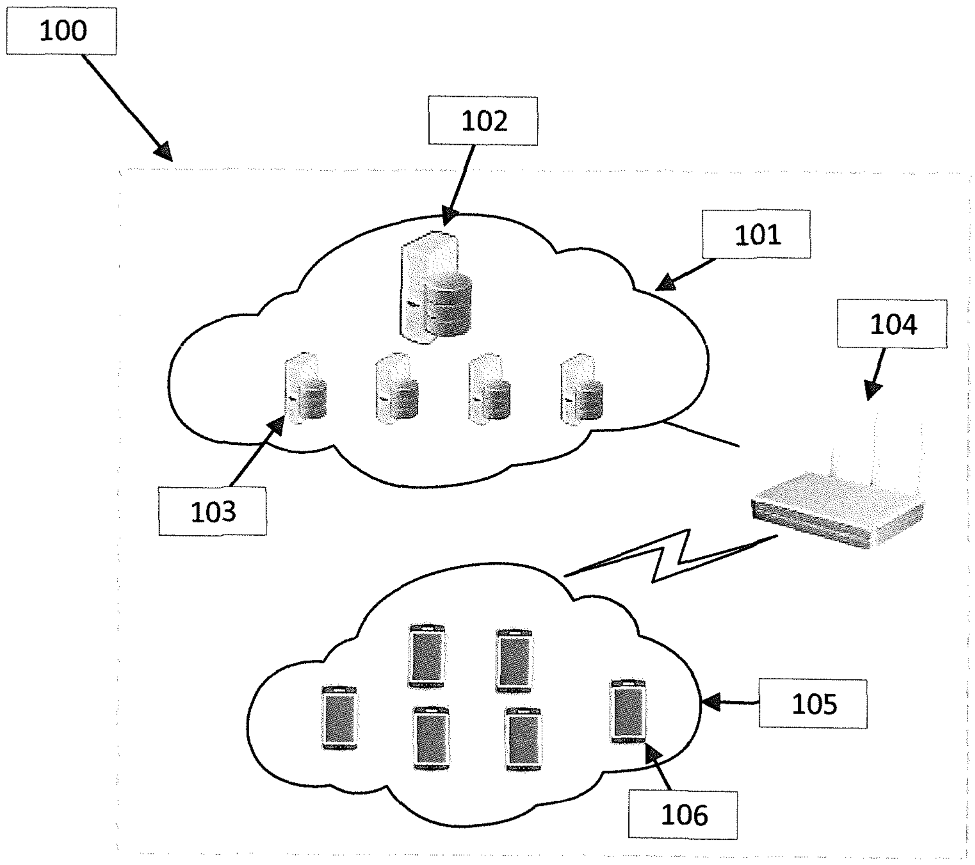

FIG. 1

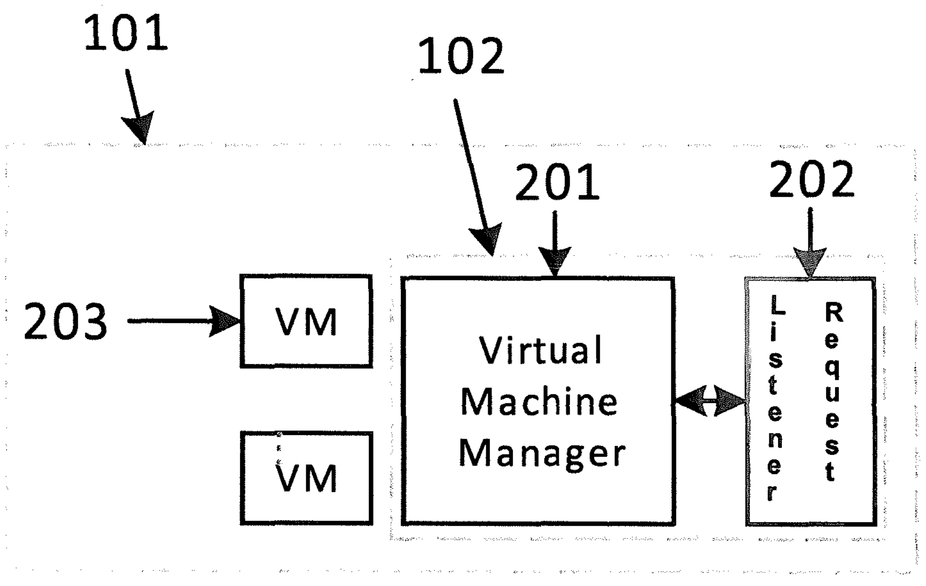
101
102
201
202
203
VM
VM
Virtual Machine Manager
Listener
Request

FIG. 2

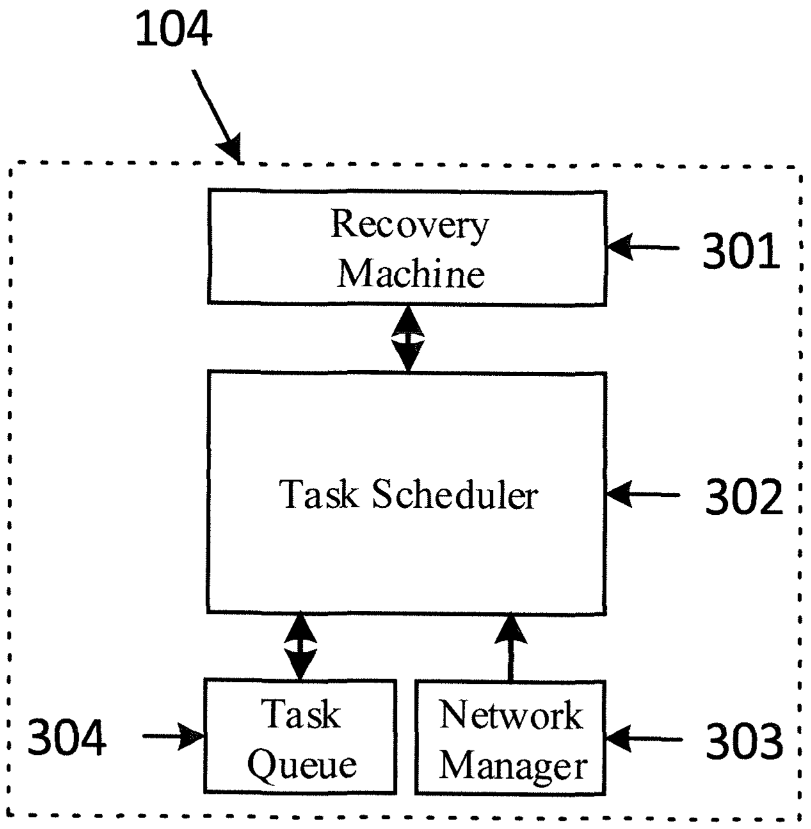
104
Recovery Machine
301
Task Scheduler
302
304
Task Queue
Network Manager
303

FIG. 3

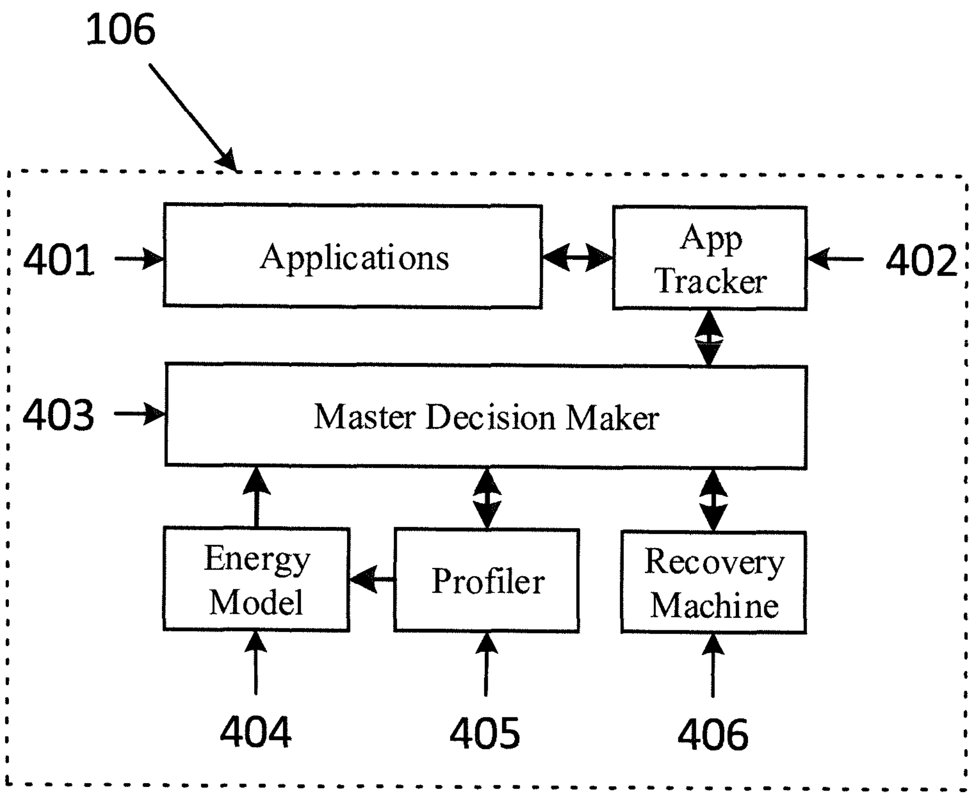
106
401
Applications
App Tracker
402
403
Master Decision Maker
Energy Model
Profiler
Recovery Machine
404
405
406

FIG. 4

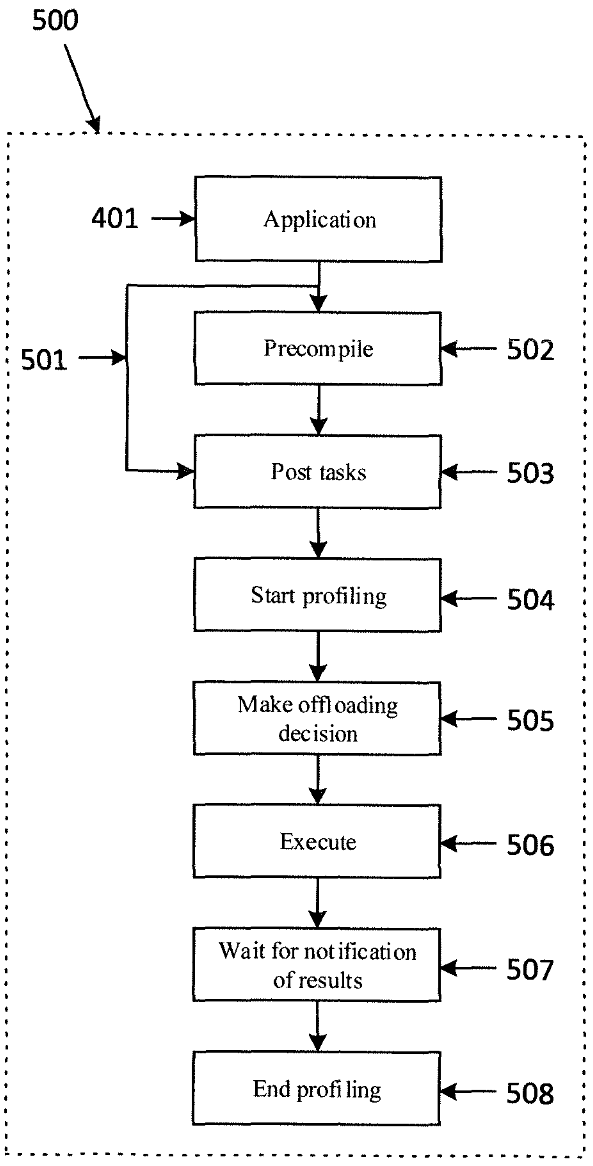
500
401
Application
501
Precompile
502
Post tasks
503
Start profiling
504
Make offloading decision
505
Execute
506
Wait for notification of results
507
End profiling
508

FIG. 5

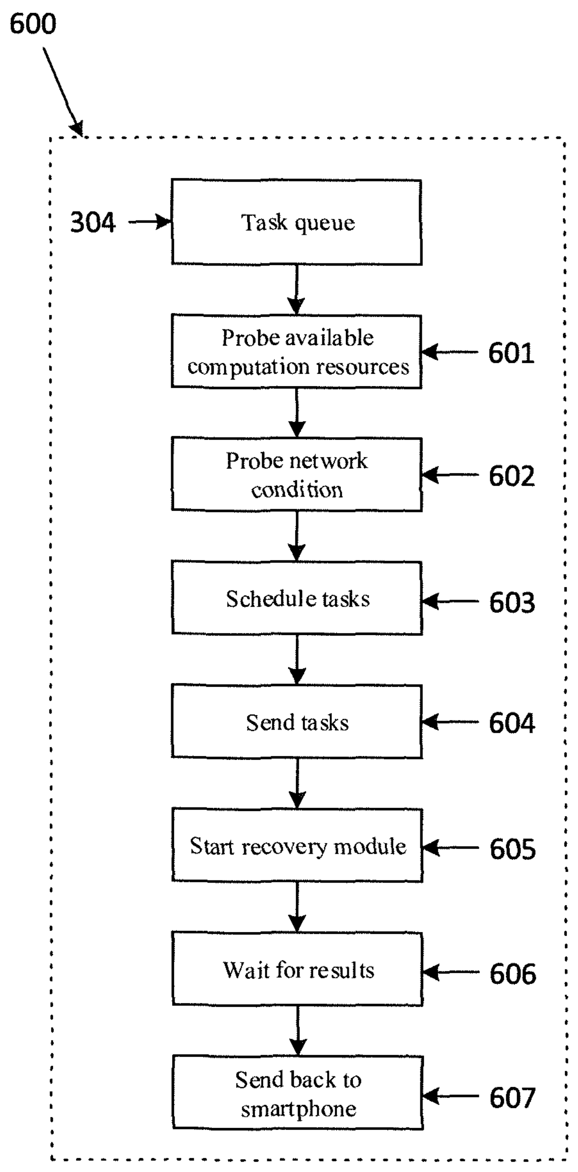
600
304
Task queue
Probe available computation resources
601
Probe network condition
602
Schedule tasks
603
Send tasks
604
Start recovery module
605
Wait for results
606
Send back to smartphone
607

FIG. 6

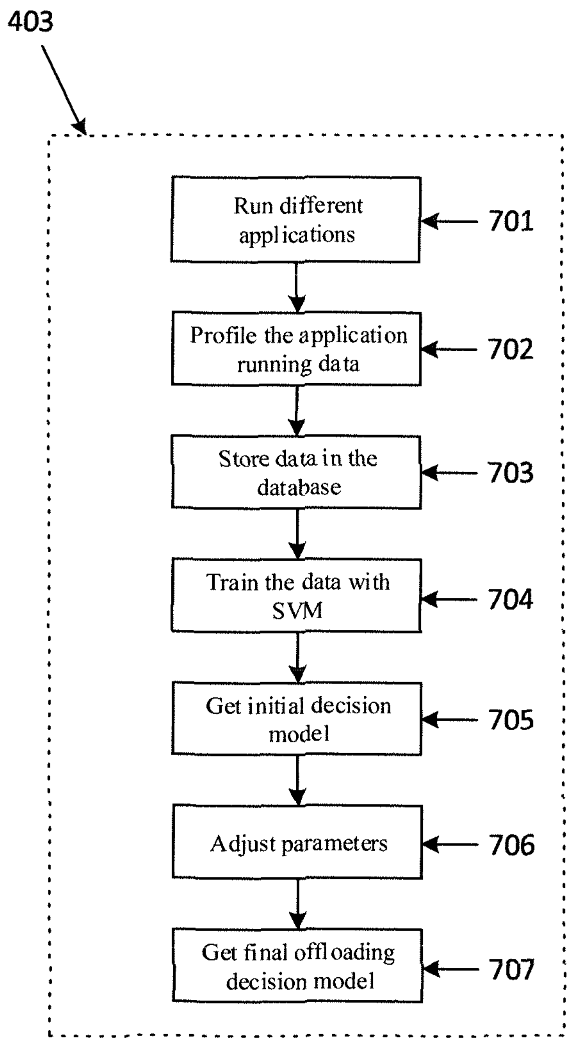
403
Run different applications
701
Profile the application running data
702
Store data in the database
703
Train the data with SVM
704
Get initial decision model
705
Adjust parameters
706
Get final offloading decision model
707

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2015/074319

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/50
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	BOWEN ZHOU ET AL: "A Context Sensitive Offloading Scheme for Mobile Cloud Computing Service", 2015 IEEE 8TH INTERNATIONAL CONFERENCE ON CLOUD COMPUTING, 27 June 2015 (2015-06-27), - 2 July 2015 (2015-07-02), pages 869-876, XP055269353, DOI: 10.1109/CLOUD.2015.119 ISBN: 978-1-4673-7287-9 figure 1; table 1 abstract page 869, left-hand column, line 25 - right-hand column, line 24 page 869, right-hand column, line 32 - page 870, left-hand column, line 4 page 870, left-hand column, line 30 - page 871, right-hand column, line 16 page 872, left-hand column, line 14 - right-hand column, line 15 -/--	1-14

[X] Further documents are listed in the continuation of Box C. [X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search 9 June 2016	Date of mailing of the international search report 20/06/2016
Name and mailing address of the ISA/ European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Authorized officer Wirtz, Hanno

1

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2015/074319

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
	page 873, left-hand column, line 4 - line 9 page 874, left-hand column, line 18 - line 37 -----	
A	US 2013/205158 A1 (LIN YING-DAR [TW] ET AL) 8 August 2013 (2013-08-08) figures 1-3 paragraph [0005] - paragraph [0007] paragraph [0011] - paragraph [0016] paragraph [0038] - paragraph [0049] -----	1-14
A	US 2013/290755 A1 (WOLMAN ALASTAIR [US] ET AL) 31 October 2013 (2013-10-31) figures 1, 3, 5-8 paragraph [0006] - paragraph [0010] paragraph [0025] - paragraph [0029] paragraph [0034] - paragraph [0036] paragraph [0044] - paragraph [0054] -----	1-14

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2015/074319

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2013205158 A1	08-08-2013	TW 201333839 A	16-08-2013
		US 2013205158 A1	08-08-2013
US 2013290755 A1	31-10-2013	US 2011231469 A1	22-09-2011
		US 2013290755 A1	31-10-2013